Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klemen Klanjšček

Aritmetične operacije v logaritemskem številskem sistemu

diplomsko delo

interdisciplinarni univerzitetni
študijski program prve stopnje
računalništvo in matematika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Raziščite logaritemski številski sistem (LNS) kot alternativo plavajoči vejici (FL). Poudarek naj bo na binarni reprezentaciji števil, implementaciji aritmetike v logiki, numeričnih napakah pri operacijah in smiselnosti uporabe LNS. Predpostavite, da znamo dobro implementirati pretvorbo iz LNS v FP in obratno.

Zahvaljujem se mentorju izr. prof. dr. Patriciu Buliću za strokovno pomoč, dostopnost in usmerjanje pri pisanju diplomske naloge. Zahvaljujem se staršem za pomoč in podporo med študijem.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
LNS	logarithmic number system	logaritmični številski sistem
VHDL	very high speed integrated circuit hardware description language	
IEEE	institute of electrical and electronics engineers	inštitut inženirjev elektrotehnike in elektronike
FP	floating point	plavajoča vejica
NaN	not a number	
XOR	exclusive disjunction	ekskluzivna disjunkcija
ROM	read only memory	bralni pomnilnik
FGPA	field-programmable gate array	
LUT	look up table	
LOD	leading one detector	detektor vodilne enice
JPEG	joint photographic experts group	
RGB	red green blue	
YCrCb	luma blue-difference red-difference	
DCT	discrete cosine transform	diskretna kosinusna transformacija
IDCT	inverse discrete cosine transform	inverzna diskretna kosinusna transformacija
BTFP	better than floating point	

Povzetek

Naslov: Aritmetične operacije v logaritemskem številskem sistemu

Avtor: Klemen Klanjšček

Namen te diplomske naloge je raziskovanje logaritmičnega številska sistema (LNS) kot alternativa plavajoči vejici (FL). Zanimale nas so binarna reprezentacija, implementacija aritmetike v logiki, numerične napake pri aritmetični operacijah in smiselnost uporabe LNS. Omejili smo se na 16-bitni LNS v bazi e. Z uporabo teorije iz enakomerne aproksimacije s polinomi smo implementirali model za aproksimacijo Gaussovih logaritmov, ki jih potrebujemo pri seštevanju in odštevanju. Aritmetično enoto v 16-bitnem LNS v bazi e smo na koncu implementirali v FPGA. Pri predpostavki, da znamo dobro implementirati pretvorbo iz LNS v FP in obratno, lahko na tak način z manjšimi spremembami implementirano aritmetično enoto uporabimo za sisteme, ki so numerično zahtevni, a ne potrebujejo natančnih končnih izračunov.

Ključne besede: aritmetika, LNS, FP, polinom, aproksimacija, enakomerna aproksimacija, Remezov algoritem, digitalno načtrovanje, FPGA.

Abstract

Title: Logarithmic Number System Based Arithmetic Operations

Author: Klemen Klanjšček

In this thesis, we discuss the logarithmic number system and its applications as an alternative to FP. We were interested in its binary representation, logical implementation of arithmetic and the numerical errors associated with it, and the arguments we can make in its favor. In particular, we focused on 16-bit LNS in base e. Using the theory of uniform approximation, a model was built for the approximation of the Gaussian logarithms needed to add and subtract in LNS. Finally, the arithmetic unit in 16-bit LNS in base e was implemented in FPGA. Assuming there is an efficient LNS to FP (and vice versa) conversion method available, we believe our implementations have applications in numerically demanding systems that do not require a high degree of accuracy in their calculations.

Key words: arithmetic, LNS, FP, polynomial, approximation, uniform approximation, Remez algorithm, digital design, FPGA.

Poglavje 1

Uvod

Namen te diplomske naloge je raziskovanje logaritmičnega številska sistema (LNS) kot alternativa plavajoči vejici (FL). Zanimale nas bodo binarna reprezentacija, implementacija aritmetike v logiki, numerične napake pri aritmetični operacijah in smiselnost uporabe LNS.

Temelje za seštevanje in odštevanje v LNS sta postavila matematika Carl Friedrich Gauss ter Zecchini Leonelli [12]. Izdala sta tabele za računaje vsote in razlike logaritmov [12].

V preteklosti in danes se LNS uporablja redko in ponavadi za zelo specializirane namene. Tak primer je superračunalnik Gravity Pipe, ki je uporabljal LNS pri simulacijah v astrofiziki [2, 21, 20]. Posluževal se je enostavnega potenciranja ($x^{\frac{3}{2}}$) v LNS [2]. LNS srečujemo pri digitalnem procesiranju signalov [17].

Glede namena uporabe aritmetične enote v LNS lahko delimo na dve smeri. V eni izmed smereh LNS primarno uporabljamo za implementacijo aritmetičnih enot BTFP (Better than Floating Point). Namen druge smeri je hitra in energijsko učinkovita implementacija manj natančnih aritmetičnih enot.

V začetnih poglavjih bomo spoznali splošno o aritmetiki v LNS in FP. Ker se LNS uporablja redko in za zelo različne namene, ni nobenih standardov, ki bi določali reprezantacijo ter aritmetiko v LNS, kot jih poznamo v FL. Zato

se bomo nadaljevanju, na podlagi nam smiselnih razlogov, omejili na 16-bitni LNS v bazi e. V takem okolju bomo definirali reprezentacijo in aritmetiko številskega sistema. Kasneje si bomo pogledali različne načine aproksimacije pri evalvaciji seštevanja in odštevanja v LNS.

Na koncu bomo tako definirano enoto implementirali z uporabo FPGA v logiki in si pogledali nekaj testov uporabe. V zaključku bomo navedli ugotovite in možnosti za nadaljnje izboljšave.

Poglavje 2

Predstavitev realnih števil v LNS

2.1 Binarna predstavitev realnih števil

Za predstavitev približkov realnih števil imamo več možnosti. Dva razširjena načina sta:

- predstavitev v fiksni vejici in
- predstavitev v plavajoči vejici.

Za predstavitev v fiksni vejici ni nobenega razširjenega standarda. Ideja je samo v tem, da bite ki, jih uporabljamo za predstavitev števila, razdelimo na bite, ki so namenjeni predstavitev celega dela, in bite, ki so namenjeni za predstavitev deljenega dela števila.

V plavajoči vejici poznamo več standardov v IEEE 754 [15, 16]. Najbolj razširjeni so:

- polovična natančnost (Half-precision), uporablja 16 bitov,
- minifloat, uporablja 24 ali manj bitov,
- enojna natančnost (Single-precision), uporablja 32 bitov, in
- dvojna natančnost (Double-precision), uporablja 64 bitov.

2.1.1 Polovična natančnost

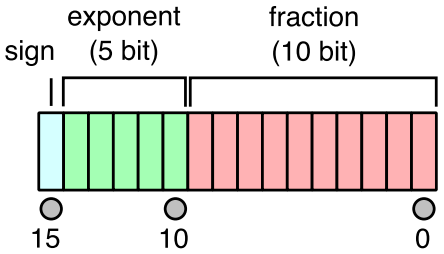
Polovična natančnost v IEEE 754 uporablja za predstavitev približkov naslednji format [15, 3]:

- Bit 15: predznak S.
- Biti od 10 do 14: eksponent E.
- Biti od 0 do 9: mantisa M.

Vrednost zapisa v polovični natančnosti je definirana v Tabeli 2.1.

E	$S = 0$	$S \neq 0$	vrednost
00000_2	± 0	subnormalizirano št.	$(-1)^S 2^{-14} \times 0.M_2$
$00001_2, \ldots, 11110_2$	normalizirano število		$(-1)^S 2^{E-15} \times 1.M_2$
11111_2	$\pm\infty$	NaN	

Tabela 2.1: Vrednosti zapisov v 16-bitnem FP.



Slika 2.1: Namembnost bitov v polovični natančnosti [10].

2.1.2 Zaokrožitvena napaka pri polovični natančnosti

Zgornja meja za relativno zaokrožitveno napako po IEEE 754 pri normaliziranih številih je $u = \frac{1}{2}2^{-10}$. Za subnormalizirana in števila izven intervala [-65504, 65504] napaka ni omejena [15, 3].

2.2 Logaritmični številski sistem

Logaritmični številski sistem LNS (Logaritmic number system) je številski sistem, v katerem so števila predstavljena z logaritmom.

Naj bo x realno število, potem par $(s, m = \log_b |x|)$ predstavlja število x v LNS z bazo b, kjer s predstavlja predznak ($s = 0$ ko je $x \geq 0$, v nasprotnem primeru je $s = 1$) in m logaritem z bazo b absolutne vrednosti x [6].

Za prestavitev realnih števil v LNS se bomo omejili na 16 bitov (glej 2.2.4). Pri LNS-ju ni nobenega standarda, zato si format predstavitve izmislimo sami.

2.2.1 Izbira baze

Smiselno je izbrati bazo, večjo od ena, saj ko je baza med nič in ena, logaritem samo spremeni predznak. Večjo bazo kot bomo izbrali, večji interval realih števil bomo pokrili z večjo relativno napako približka. Pri predstavitvi števila v LNS z bazo e se njena relativna napaka zanemarljivo poveča v primerjavi z analogom v FP (glej 2.2.4). Izbrali smo bazo e.

2.2.2 Dodeljevanje bitov

Prvi bit predstavlja predznak s. Ostali biti predstavljajo logaritem m. Za predstavitev logaritma bomo izbrali fiksno vejico (glej 2.2.4). Ker je logaritem lahko negativen, bomo za predstavitev, zaradi lažje implementacije, uporabili dvojiški komplement. Lahko bi hranili posebej bit za predznak in absolutno vrednost logaritma. Pri postavitvi fiksne vejice logaritma v LNS si pomagamo s polovično natančnostjo (glej 2.2.4). Na podlagi prejšnjih ugotovitev smo za predstavitev števil v 16-bitnem LNS definirali naslednji format.

- Bit 15: predznak s.
- Biti od 0 od 14: logaritem m v dvojiškem komplementu.

- Eksponent p je fiksen z vrednostjo -10.

Vrednost zapisa v tako definiranem LNS je $(-1)^s e^{m\,2^p}$.

Fiksna vejica nam omogoča, da z logaritmom računamo kot s predznačenim celim številom.

2.2.3 Prekoračitve in podkoračitve

V LNS število nič ni predstavljivo, ker ni v definicijskem območju logaritma. V našem primeru za števila, ki so izven intervala $[-e^{16-2^{-9}}, -e^{-16+2^{-10}}] \cup [e^{-16+2^{-10}}, e^{16-2^{-9}}]$, s tabelo 2.2 definiramo prekoračitve in podkoračitve.

interval	s	m	število
$[\exp(16-2^{-10}), \infty)$	0	$16-2^{-10}$	∞
$[0, \exp(-16)]$	0	-16	0
$(-\infty, -\exp(16-2^{-10})]$	1	$16-2^{-10}$	$-\infty$
$[-\exp(-16), 0]$	1	-16	-0

Tabela 2.2: Definicije prekoračitev in podkoračitev v 16-bitnem LNS.

V LNS nimamo števil, kot so subnormalizirana števila pri plavajoči vejici. Ta števila se LNS zaokrožijo v LNS(0) oz. LNS(-0).

2.2.4 Zaokrožitvena napaka

Pri predpostavki, da logaritem števila x na intervalu $[\min m, \max m]$ zaokrožujemo z absolutno napako δ, je relativna zaokrožitvena napaka u števila x v LNS pri bazi b:

$$\mathrm{LNS}(x) = b^{\log_b x+\delta} = xb^{\delta} = x(1+\ln(b)\delta+\mathcal{O}(\delta^2)) = x(1+u) \tag{2.1}$$

V izpeljavi 2.1 opazimo, da je v LNS relativna zaokrožitvena napaka u skoraj linearno odvisna ($|u| \approx \ln(b)|\delta|$) od absolutne zaokrožitvene napake δ logaritma x. To pomeni, da lahko logaritem števila x predstavimo s

fiksno vejico, ki ima absolutno zaokrožitveno napako. Zgornja meja za absolutno zaokrožitveno napako δ pri pravilnem (zaokrožujemo k najbližjemu predstavljivemu številu) zaokroževanju logaritma števila x v fiksni vejici z eksponentom p je 2^{p-1}.

Če pri 16-bitnem LNS izberemo bazo e in eksponent $p = -10$, bo relativna zaokrožitvena napaka približno enaka polovični natančnosti, interval predstavljivih števil v LNS pa večji od intervala normaliziranih števil v polovični natančnosti. Če pri 16-bitnem LNS izberemo bazo 2 in eksponent $p = -10$, bo relativna zaokrožitvena napaka manjša od polovične natančnosti, interval predstavljivih števil v LNS pa približno enak normaliziranih števil v polovični natančnosti.

Poglavje 3

Aritmetika v splošnem LNS

3.1 Množenje in deljenje

Naj bosta $x = (s_x, m_x)$ in $y = (s_y, m_y)$ dve števili v LNS, potem se množenje xy prevede na seštevanje $xy = (s_x + s_y \mod 2, m_x + m_y)$ in deljenje $\frac{x}{y}$ na odštevanje $\frac{x}{y} = (s_x + s_y \mod 2, m_x - m_y)$. Deljenje in množenje z nič ne obstaja, ker nič ni predstavljivo število.

3.2 Potenciranje in korenjenje

Naj bo $x = (0, m)$ pozitivno število v LNS, potem se potenciranje z realnim številom r prevede na množenje $x^r = (0, mr)$. Podobno velja za korenjenje.

3.3 Seštevanje in odštevanje

Naj bosta $x = b^u$ in $x = b^v$ dve števili v LNS. Brez izgube splošnosti lahko predpostavimo, da sta x in y pozitivni števili in da $x \geq y$. Računajmo:

$$x + y = b^u + b^v = b^u(1 + b^{(v-u)}) = b^{(u+\log_b(b^{(v-u)}+1))} \tag{3.1}$$

$$x - y = b^u - b^v = b^v(b^{(u-v)} - 1) = b^{(v+\log_b(b^{(u-v)}-1))} \tag{3.2}$$

V izpeljavi 3.1 in 3.2 vidimo, da bo pri seštevanju in odštevanju potreben izračun t. i. Gaussovih logaritmov [13], in sicer sumacijske $\mathrm{sum}_b(z) = \log_b(b^z + 1)$ in diferenčne $\mathrm{diff}_b(z) = \log_b(b^z - 1)$ funkcije.

Poglavje 4

Aritmetika v 16-bitnem LNS v bazi e

4.1 Množenje in deljenje

Računanje predznaka ne predstavlja problema (ekskluzivna disjunkcija). Pri seštevanju oz. odštevanju 15-bitnih števil v dvojiškem komplementu lahko pride do preliva. Ko pride do preliva, je rezultat prekoračitev ali podkoračitev. Ker v 16-bitnem LSN nimamo predstavitve za NaN (Not a Number), se ilegalne operacije (deljenje z nič) evalvirajo, kot če bi bile operacije nad predstavljivimi števili. Primeri takih operacij so v tabeli 4.1.

V taki implentaciji rezultat množenja z LNS(0) oz. deljenja z LNS(∞) ni nujno enak LNS(0). Za implementacijo pravilnega množenja z LNS(0) (rezultat množenja z LNS(0) je LNS(0)) je potrebna dodatna logika.

vhod	rezultat	opomba
0×0	0	pride do preliva
$0 \times \infty$	$e^{-2^{-10}} \approx 1$	
$\infty \times \infty$	∞	pride do preliva
$0 \div 0$	1	
$0 \div \infty$	0	pride do preliva
$\infty \div \infty$	1	
$\infty \div 0$	∞	pride do preliva

Tabela 4.1: Operacije z nepredstavljivimi števili.

4.2 Potenciranje in korenjenje

Računaje inverza se v LNS prevede na spremenjenje predznaka logaritma. Potenciranje z 2^k se v LNS prevede na aritmetični pomik logaritma v desno oz. levo glede na predznak števila k. Pri pomiku v desno bo potrebno zaokroževanje. Če želimo potencirati z nekim številom v fiksni vejici (smiselna velikost števila je do 15 bitov v fiksni vejici), bomo morali opraviti množenje in upoštevati možnost preliva. (Opomba: potenciramo s številom, ki ni v predstavljeno v LNS.)

4.3 Seštevanje in odštevanje

Pri seštevanju in odštevanju v bazi e bo potrebna dovolj natančna evalvacija funkcij $g_1(z) = \ln(e^z + 1)$ $g_2(z) = \ln(e^z - 1)$. V podpoglavju 3.3 smo predpostavili, da sta argumenta $x = (s_x, u)$ in $y = (s_y, v)$ nenegativna in da velja $|x| \geq |y|$. V splošnem to ni res, zato bomo vsoto oz. razliko računali na način, ki je predpisan s tabelo 4.2.

predznak x	predznak y	operacija	$\lvert x\rvert \geq \lvert y\rvert$	rezultat
+	+	+	da	$\exp(v + g_1(u - v))$
+	-	+	da	$\exp(v + g_2(u - v))$
-	+	+	da	$-\exp(v + g_2(u - v))$
-	-	+	da	$-\exp(v + g_1(u - v))$
+	+	-	da	$\exp(v + g_2(u - v))$
+	-	-	da	$\exp(v + g_1(u - v))$
-	+	-	da	$-\exp(v + g_1(u - v))$
-	-	-	da	$-\exp(v + g_2(u - v))$
+	+	+	ne	$\exp(u + g_1(v - u))$
+	-	+	ne	$-\exp(u + g_2(v - u))$
-	+	+	ne	$\exp(u + g_2(v - u))$
-	-	+	ne	$-\exp(u + g_1(v - u))$
+	+	-	ne	$-\exp(u + g_2(v - u))$
+	-	-	ne	$\exp(u + g_1(v - u))$
-	+	-	ne	$-\exp(u + g_1(v - u))$
-	-	-	ne	$\exp(u + g_2(v - u))$

Tabela 4.2: Seštevanje in odštevanje pri splošnih argumentih.

4.4 Analiza napak pri aritmetiki v 16-bitnem LNS

Pri predpostavki, da so argumenti točni in je rezultat predstavljivo število (ne pride do preliva), imamo naslednje napake.

Pri množenju in deljenju ni napake, saj seštevamo oz. odštevamo dve števili v fiksni vejici. Računanje inverza je brez napake, saj spremenimo samo predznak. Potenciranje s celim številom je brez napake, saj ne dobimo nobenih novih deljenih bitov. Pri potenciranju oz. korenjenju s poljubnim številom v fiksni vejici je zgornja meja za napako $|\delta| \leq 2^{-11}$, saj zaokrožimo logaritem na najbližje predstavljivo število v 15-bitni fiksni vejici.

4.4.1 Napaka pri seštevanju oz. odštevanju

Pri polovični natančnosti je pri vseh aritmetičnih operacijah zgornja meja za relativno napako, pri predpostavki, da so argumenti točni in normalizirani, 2^{-11} (po IEEE 754).

Vse napake v LNS od tukaj naprej gledamo kot absolutne (v bazi e je $\delta \approx u$). Pri seštevanju oz. odštevanju v 16-bitnem LNS imamo dve vrsti napake:

- napaka E_m pri aproksimaciji in evalvaciji funkcij g_1 in g_2 ter
- končna zaokrožitvena napaka E_z, rezultat seštevanja oz. odštevanja zaokrožimo.

Na napako $|E_z| \leq 2^{-11}$ ne moremo vplivati (lahko bi zmanjšali eksponent p, kar bi zmanjšalo interval predstavljivih števil). Napaka E_m je lahko poljubno majhna (vsaj v teoriji). Končna napaka je torej $E = |E_m| + |E_z|$. V bazi e ne moremo doseči enake ali manjše zgornje meje za relativno napako pri seštevanju in odštevanju kot pri analogu v FL, saj bi napaka E_m morala biti enaka nič (glej 2.2.4). V bazi $b < e$ se da doseči manjšo relativno napako pri seštevanju in odštevanju kot pri analogu v FL, če je $|E_z| \leq 2^{-11}(\frac{1}{\ln b} - 1)$.

Lahko dosežemo, da je napaka pod $|E_z| + \epsilon$. Manjši ϵ bomo izbrali natančneje oz. težje bo aproksimirat g_1 in g_2. Določene aplikacije se ne zanašajo na strogo zgornjo mejo za napako pri seštevanju in odštevanje (računalniška grafika, kodiranje in dekodiranje zvoka, filmov, slik), zato lahko izberemo $\epsilon = 2^{-11}$, kar nam da skupno napako $|E| \leq 2^{-10}$. To pomeni, da bodo vsi biti točni, razen včasih (v praksi redko) zadnji LSB (Least Significant Bit) bit.

Poglavje 5

Aproksimacija in evalvacija funkcij g_1 ter g_2

5.1 Interval evalvacije

Funkcija g_1 ima za definicijsko območje celo realno os. Brez problema lahko območje evalvacije skrčimo na pozitivna oziroma negativna števila (glej 4.3). Aproksimacija in evalvacija sta težji za negativni del, zato se omejimo na pozitivnega.

Če seštevamo oz. odštevamo dve števili, katerih absolutna razlika je velika, je rezultat zaradi zaokroževanja kar največji vhodni argument. Pri funkciji se g_1 se to vidi v limiti:

$$\lim_{z\to\infty} g_1(z) - z = \lim_{z\to\infty} \ln(e^z + 1) - z = 0$$

Ker je $g_1(z) - z$ monotono padajoča funkcija, velja, da je za $z \gtrapprox 7.6243$ napaka aproksimacije funkcije g_1 s funkcijo $\mathrm{id}(z) = z$ manjša od 2^{-11}. Razlog, zakaj smo izbrali napako 2^{-11} in ne 2^{-10}, je, da je v praksi skupna napaka aproksimacije in evalvacije manjša od 2^{-10}. Poleg tega izbira napake 2^{-10} bistveno ne zmanjša intervala (6.9309, kar se zaokroži na 7). V praksi bomo zaradi lažje implementacije funkcijo g_1 aproksimirali samo na intervalu $[0, 8)$.

Definicijsko območje funkcije g_2 so pozitivna realna števila. Podobno kot

pri g_1 je meja za funkcijo g_2 približno enaka 7.6249. Zaradi lažje implementacije interval zaokrožimo na $[0, 8)$.

5.2 Aproksimacija

Funkciji g_1 in g_2 bi lahko evalvirali brez aproksimacije, tako da bi si za vse vhode ($8 \times 2^{10} = 8192$) rezultate hranili v ($8K \times 13$) velikem ROM-u za vsako funkcijo posebej. Namesto ROM-a bi lahko funkciji implementirali direktno s kombinacijsko logiko. Rešitvi sta v praski nepraktični, ker porabimo zelo veliko prostora (logičnih vrat).

Porabo prostora bomo zmanjšali z aproksimacijo funkcij g_1 in g_2 s polinomi (glej 6).

5.2.1 Aproksimacija funkcije g_1

Funkcija g_1 je na intervalu $[0, 8)$ gladka ($\in \mathcal{C}^\infty$), zato aproksimacija s polinomom ni problematična.

5.2.2 Aproksimacija funkcije g_2

Funkcija g_2 je na intervalu $[0, 8)$ gladka ($\in \mathcal{C}^\infty$), pri 0 pa ima singularnost. Ker napaka aproksimacije s polinomi v bližini singularnosti hitro raste, je aproksimacija funkcije na intervalu $(0, \gamma)$ v trenutni obliki nepraktična.

Kritične dele funkcije g_2 obravnavamo posebej:

- $\{0\}$ – Singularnost obravnavamo posebej (vrednost v singularnosti je v našem primeru LNS($-\infty$)).
- $(0, \gamma)$ – Funkcijo razdelimo tako, da se znebimo vpliva singularnosti.
- $[\gamma, 8)$ – Funkcijo evalviramo kot g_2.

Na intervalu $(0, \gamma)$ na naslednji način zmanjšamo vpliv singularnosti:

$$g_2(z) = \ln(e^z - 1) = \ln(\frac{e^z - 1}{z} z) = \ln(\frac{e^z - 1}{z}) + \ln z$$

Tako smo funkcijo g_2 prevedli na kompozicijo funkcij $g_3(z) = \ln(\frac{e^z-1}{z})$ in $g_4(z) = \ln z$. Kako izberemo γ, je odvisno od aproksimacijske metode.

5.2.3 Aproksimacija funkcije g_3

Funkcija $g_3(z)$ pri nič nima singularnosti.

$$\lim_{z\to 0} \ln(\frac{e^z-1}{z}) = \ln(\lim_{z\to 0} \frac{e^z-1}{z}) = \ln 1 = 0$$

Funkcija g_3 je vsaj enkrat zvezno odvedljiva. Aproksimacija s polinomom ni problematična.

5.2.4 Aproksimacija funkcije g_4

Funkcija g_4 ima singularnost pri nič, kar pomeni, da aproksimacija s polinomom na $(0, \gamma)$ ni praktična. Z uvedbo novih spremenljivk interval evalvacije zmanjšamo na $[1, 2)$ in se oddaljimo od singularnosti. Računajmo:

$$g_4(z) = \ln z = \ln(z2^h2^{-h}) = \ln(z2^h) + \ln(2^{-h}) = \ln(z2^h) - h\ln 2$$

Ker je z predstavljen v fiksni vejici, bo le potrebna implementacija logike za aritmetični levi pomik in za izračun vodilne enice.

5.3 Evalvacija

Polinome bomo evalvirali s Hornerjevo metodo. Zaradi lažje implementacije in vzdrževanja napake (napaka pri zaokroževanju koeficientov (fiksna vejica) in napaka pri množenju) bomo pri evalvaciji polinomov uvedli novo spremenljivko, tako da bo območje evalvacije na intervalu $[0, 1)$.

Naj bo $p(z) = \sum_{i=0}^{k} a_i z^i$ polinom stopnje k s koeficienti $(a_i)_{i=0}^{k}$, potem je oblika 5.1

$$p(z) = ((((a_k z + a_{k-1})z + a_{k-2})z + a_{k-3})z + \cdots + a_0) \tag{5.1}$$

evalvacija polinoma p v točki z s Hornerjevo metodo.

Ker računamo v fiksni vejici, je napaka evalvacije polinoma p na intervalu $[0, 1)$ enaka $E_p \leq \sum_{i=0}^{k} E_{a_i} + \sum_{i=1}^{k} E_{m_i}$, pri čemer je E_{a_i} zaokrožitvena napaka koeficienta a_i in E_{m_i} napaka pri i-tem množenju v Hornerjevi metodi.

Skupna napaka E_m je vsota napake E_a pri aproksimacijski metodi in napake E_e pri evalvaciji polinoma. Želimo, da skupna napaka ne presega 2^{-11}.

Poglavje 6

Teorija aproksimacije

Funkcijo g želimo na diskretnem območju $[a, b] \cap D$ (D je množica predstavljivih števil v 16-bitnem LNS) aproksimirati s polinomom p, ki bo zadostoval naslednjemu pogoju:

$$\max_{z \in [a,b] \cap D} |g(z) - p(z)| \leq E,$$

pri čemer je E zgornja meja za aproksimacijsko napako. Vsi izreki in njihovi dokazi so v delih [18, 19].

Izrek 6.1 *(Weierstrass). Če je funkcija f zvezna na $[a, b]$, potem za poljuben $\epsilon > 0$ obstaja tak polinom p, da je:*

$$|f(x) - p(x)| \leq \epsilon \text{ za vsak } x \in [a, b].$$

Če označimo $dist(f, P_n) := \min_{p \in P_n} \|f - p\|_\infty$, potem iz izreka sledi

$$\lim_{k \to \infty} dist(f, P_k) = 0.$$

Potemtakem, ker je $[a, b] \cap D \subset [a, b]$, iz Weierstrass sledi, da tak polinom p za funkcijo g obstaja.

Želimo poiskati polinom $p \in P_n$, čim nižje stopnje, ki bo zadostoval omejitvam napake. Iskanje optimalnega polinoma $p \in P_n$ nad diskretno množico $[a, b] \cap D$ je minimax optimizacijski problem (diskretna najboljša enakomerna aproksimacija):

$$\min_{p\in P_k} \max_{z\in[a,b]\cap D} |g(z)-p(z)|$$

Minimalni n poiščemo tako, da n povečujemo za ena, dokler rešitev minimax problema ne zadosti omejitvam napake. Ker imamo veliko točk (do 2^{13}), je v praksi iskanje polinoma z reševanjem minimax problema počasno. Glede na to, da je g zvezno odvedljiva funkcija, je v naših primerih iskanje polinoma najboljše zvezne aproksimacije računsko lažje.

Napako lahko minimiziramo na celem intervalu $[a,b]$, namesto samo v diskretnih točkah. Polinom $p \in P_n$ zvezne najboljše enakomerne aproksimacije na $[a,b]$ minimizira $\|g-p\|_{\infty,[a,b]}$.

Trivialno velja, da če polinom p, po normi $\|\cdot\|_\infty$ na intervalu $[a,b]$ najboljše aproksimira g z napako, manjšo od E, potem bo ta polinom zadoščal tudi za diskretne točke. V praksi se izkaze, da je razlika v naših primerih zanemarljiva.

Izrek 6.2 *Naj bo f zvezna na $[a,b]$. Če za polinom p stopnje $\leq n$ velja, da razlika $f-p$ alternirajoče zavzame ekstremno vrednost na $[a,b]$ v vsaj $n+2$ točkah $a \leq x_0 < x_1 < \cdots < x_{n+1} \leq b$, tako da velja $(f(x_i)-p(x_i))(f(x_{i+1})-p(x_{i+1})) < 0$ za $i = 0,1,\ldots,n$, potem je p polinom najboljše enakomerne aproksimacije za f na $[a,b]$ v P_n.*

Dokaz.

Denimo, da p ni polinom najboljše enakomerne aproksimacije in naj bo $g \in P_n$ pravi polinom najboljše aproksimacije. To pomeni, da je $|f(x_i)-g(x_i)| < |f(x_i)-p(x_i)|$ za $i = 0,\ldots,n+1$. Trdimo, da zato velja, da je $(f(x_i)-p(x_i))-(f(x_i)-g(x_i))$ istega predznaka kot $f(x_i)-p(x_i)$ za $i = 0,\ldots,n+1$. Potem je tudi $g-p$ polinom stopnje $\leq n$, za katerega velja, da vrednost v $n+2$ točkah alternira. Iz tega sledi, da mora $g-p$ imeti $n+1$ ničel $\rightarrow\leftarrow$. p je res polinom najboljše aproksimacije. □

Algoritem 1. (Prvi Remezov postopek)

Izberi $n+2$ delilnih točk $x_0 < x_1 < \cdots < x_{n+1}$ na $[a,b]$ (točke Čebiseva so dober začetni približek) in dokler ni konvergence, ponavljaj:

- določi $p \in P_n$ in r, da velja $f(x_i) - p(x_i) = (-1)^i r$ za $i = 0, 1, \ldots, n+1$ (sistem $n+2$ linearnih enačb);
- poišči y na $[a, b]$, kjer razlika $|f(x) - p(x)|$ doseže svoj maksimum (ker $f'(x)$ in $p'(x)$ poznamo, je iskanje lokalnih ekstremov lažje);
- če je $|f(y) - p(y)| - |r| < \epsilon$, potem končaj;
- točko y zamenjaj z eno izmed točk $x_0, \ldots, x_{n+1}$ (ohranjanje alternacije):
  - če je $x_k < y < x_{k+1}$, zamenjaj y s točko x_k, če je $f(x_k) - p(x_k)$ istega predznaka kot $f(y) - p(y)$, sicer zamenjaj s x_{k+1};
  - če je $a \leq y < x_0$ in če je predznak razlike $f - p$ v y enak kot v x_0, zamenjaj y z x_0, sicer zamenjaj y in x_{n+1};
  - če je $x_{n+1} < y \leq b$ in če je predznak razlike $f - p$ v y enak kot v x_{n+1}, zamenjaj y in x_{n+1}, sicer y in x_0.

Lema 6.1 *Naj bo f zvezna na $[a, b]$. Če s $p \in P_n$ velja, da razlika $f - p$ alternira v $n+2$ točkah $x_0 < x_1 < \cdots < x_{n+1}$ na $[a, b]$, potem velja:*

$$|r| = \min_{0 \leq i \leq n+1} |f(x_i) - p(x_i)| \leq dist(f, P_n) \leq \|f - p\|_\infty$$

Dokaz. Naj bo $q \in P_n$ polinom najboljše enakomerne aproksimacije na $[a, b]$ za f. Desna neenakost je očitna. Denimo, da je $|f(x_i) - p(x_i)| > \text{dist}(f, P_n)$ za $i = 0, 1, \ldots, n+1$. Tako kot v dokazu izreka 6.2 sledi, da je predznak $(f - p) - (f - g)$ v točkah $x_0, \ldots, x_n$ enak predznaku $f - p$. Iz tega sledi, da mora $g - p$ imeti $n+1$ ničel $\rightarrow\leftarrow$. Prva neenakost velja. □

Za izpolnjevanje pogojev za aproksimacijsko napako je stopnja polinoma lahko visoka. Problem rešujemo tako, da interval razdelimo na podintervale in vsakega posebej aproksimiramo s svojim polinomom.

Poglavje 7

Simulacija – MATLAB

V programskem okolju MATLAB smo implementirali simulator za aritmetično enoto v 16-bitnem LNS. Aritmetična enota vsebuje naslednje operacije: množenje, deljenje, seštevanje, odštevanje, korenjenje in kvadriranje. Pri iskanju aproksimacijski polinomov smo uporabili odprtokodno knjižnico Chebfun [1]. Za reševanje minimax problemov smo uporabili funkcijo fminimax v MATLAB-u.

7.1 Aproksimacija brez segmentacije

Funkcije g_1, g_3 in g_4 bomo aproksimirali v celoti (na intervalu evalvacije) z enim polinomom.

7.1.1 Aproksimacija funkcije g_1

Ker je g_1 na intervalu $[0, 8)$ zvezno odvedljiva, jo lahko poljubno dobro aproksimiramo s polinomi. Vsota napake aproksimacije E_a in napake evalvacije E_e ne sme presegati 2^{-11}.

Želimo, da bo napaka uravnotežena, torej izberemo $E_a \leq 2^{-12}$ in $E_e \leq 2^{-12}$.

Polinom poiščemo tako, da povečujemo stopnjo polinoma, dokler ne zadostimo napaki E_a v P_i. Napako dosežemo pri polinomu $p \in P_8$ z napako

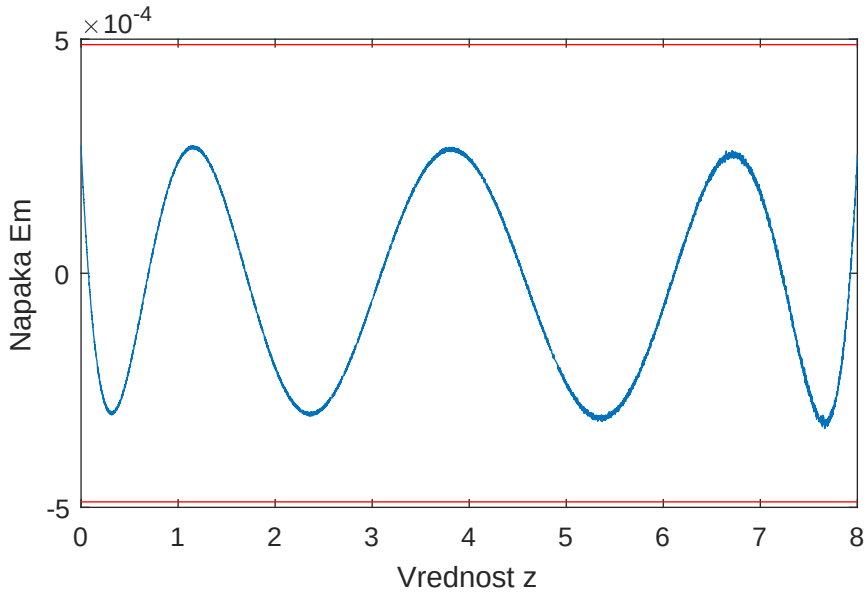
$E_a \approx 5.0517 \times 10^{-5}$. Pri $p \in P_7$ dobimo $E_a \approx 2.8410 \times 10^{-4}$.

Zaradi lažjega shranjevanja koeficientov, računanja napake in evalviranja polinoma bomo uvedli novo spremenljivko $w = \frac{z}{8}$. Funkcija g_1 z novo spremenljivko je $g_1(w) = \ln(e^{8w} + 1)$. Z uvedbo nove spremenljivke smo skrčili interval na $[0, 1)$. Pred uvedbo nove spremenljivke so bili koeficienti polinoma po absolutni vrednosti zelo različni. Po uvedbi nove spremenljivke je razmerje pri $p \in P_7$ med najmanjšim in največjim koeficientom padlo s približno 234403 na približno 45. Ko bomo polinom evalvirali s Hornejevo metodo, bomo po množenjih vedno dobili manjše število. Med drugim so v novi spremenljivki biti koeficientov za fiksno vejico enako pomembni za napako. Koeficiente polinomov bomo zaokrožili na enako bitov za fiksno vejico. Napaka pri zaokrožitvi (zaokroževanje na najbližje predstavljivo število) na t bitov za fiksno vejico je 2^{t-1}. Pri vsakem množenju odstranimo nekaj zadnjih bitov (manj bitov za naslednje množenje). Pri rezanju bitov za j-tim bitom za fiksno vejico dobimo napako 2^{-j}.

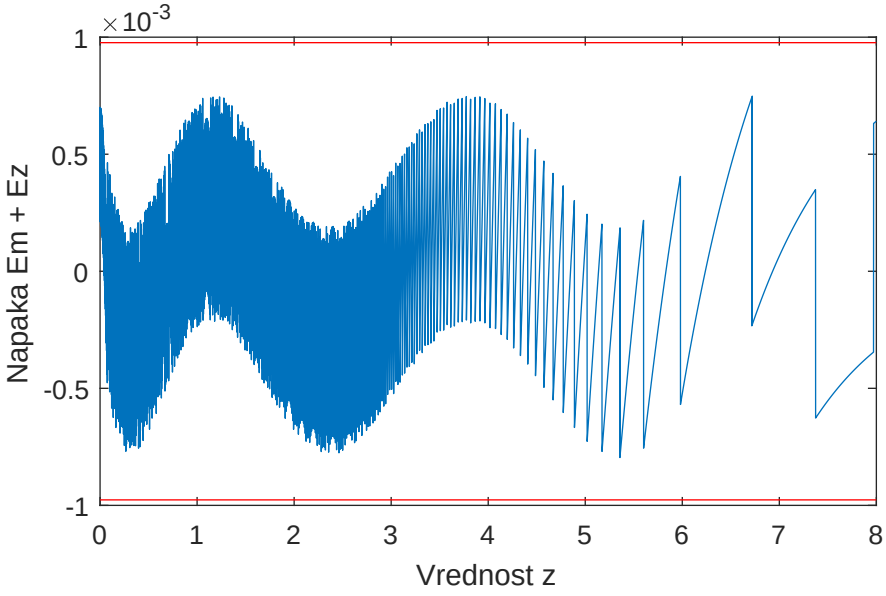
Pri izberi polinoma $p \in P_8$ zaokrožimo koeficiente na 15 bitov za fiksno vejico in pri množenjih odstranimo bite po 16. bitu za fiksno vejico, saj $|E_a + E_e| \leq 5.0517 \times 10^{-5} + 9 \times 2^{-16} + 8 \times 2^{-16} = 3.0992 \times 10^{-4} \leq 2^{-11}$.

Glede na to, da je napaka $E_a \approx 5.0517 \times 10^{-5}$ pri $p \in P_7$ dokaj blizu 2^{-12}, lahko izberemo zaokroževanje pri koeficientih in množenju tako, da je kompozicija napak manjša od 2^{-11}. Pri ohranjanju 17 bitov za fiksno vejico pri množenju in zaokroževanju na 15 bitov za fiksno vejico pri koeficientih je skupna napaka $|E_a+E_e| \leq 2.8410\times10^{-4}+8\times2^{-16}+7\times2^{-17} = 3.9092\times10^{-4} \leq 2^{-11}$. Naša končna izbira za g_1 je polinom $p \in P_7$. Na grafu 7.1 je prikazana napaka $E_m = E_a + E_e$. Rdeči črti označujeta omejitev napake z 2^{-11}. Na grafu 7.2 je prikazana napaka $E_m + E_z$ po končnem zaokroževanju. Rdeči črti označujeta omejitev napake z 2^{-10}.

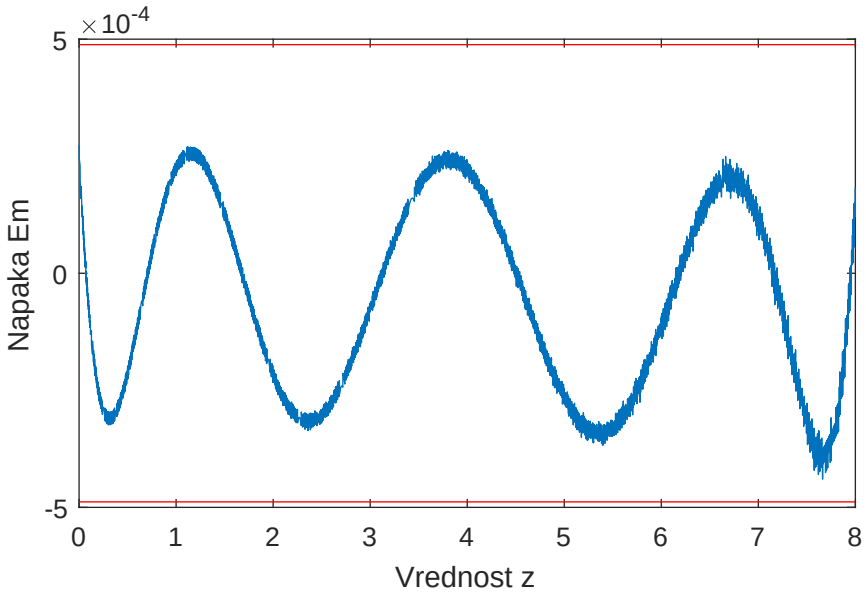
V praksi ohranjanje 15 bitov za fiksno vejico pri množenju nima bistvenega vpliva na končno napako v primerjavi z ohranjanjem 17 bitov za fiksno vejico. Spremembe v napaki se vidijo na grafih 7.3 in 7.4.



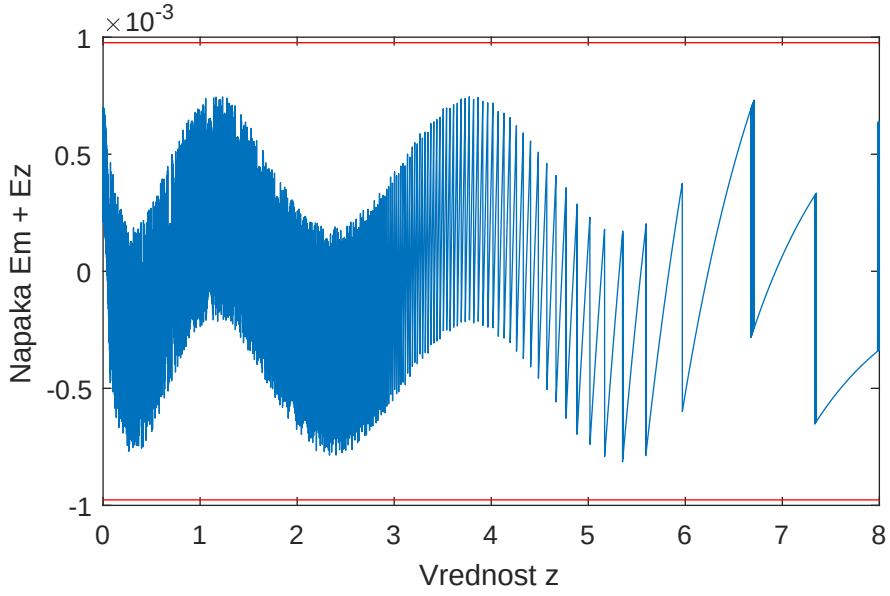
Slika 7.1: Graf napake E_m za g_1 na $[0, 8)$.



Slika 7.2: Graf napake $E_m + E_z$ za g_1 na $[0, 8)$.



Slika 7.3: Graf napake E_m za g_1 z modificiranim množenjem na $[0, 8)$.



Slika 7.4: Graf napake $E_m + E_z$ za g_1 z modificiranim množenjem na $[0, 8)$.

7.1.2 Aproksimacija funkcije g_2

Pri funkciji g_2 aproksimiramo samo funkciji g_3 in g_4, ker želimo interval $(0, 8)$ aproksimirati v celoti. Parameter γ je 8. Skupna napaka $|E_a(g_3) + E_e(g_3) + E_a(g_4) + E_e(g_4) + E_a(h \ln 2)|$ ne sme presegati 2^{-11}. Ker želimo, da bo napaka med funkcijama enakomerno utežena, poiščemo polinoma za g_3 in g_4, za katera velja $|E_a(g_3)| \leq 2^{-13}$ in $|E_a(g_4)| \leq 2^{-13}$. Iz enakih razlogov kot pri g_1 uvedemo nove spremenljivke, tako da je $g_3(u) = \ln(\frac{e^{8u}-1}{8u})$ in $g_4(v) = \ln(2v)$. Na ta način smo skrčili intervala z $(0, 8)$ in $[1, 2)$ na $(0, 1)$ in $[0.5, 1)$.

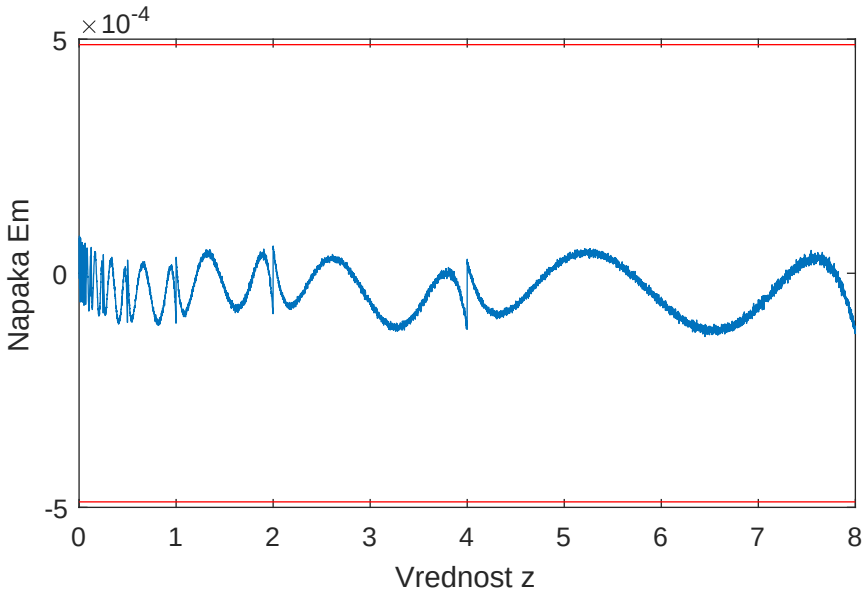
Da zadostimo napakam pri g_3, poiščemo $p \in P_6$ z napako $E_a = 1.9325 \times 10^{-05}$ in pri g_4 pa $p \in P_4$ z napako $E_a = 6.0714 \times 10^{-05}$. Da bo napaka $|E_e(g_3)| \leq 2^{-12} - |E_a(g_3)|$, ohranjamo 17 bitov za fiksno vejico pri množenju in koeficiente zaokrožimo na 15 bitov za fiksno vejico. Pri g_3 je pri taki izbiri skupna napaka $|E_a(g_3) + E_e(g_3)| \leq 1.9325 \times 10^{-5} + 7 \times 2^{-16} + 6 \times 2^{-17} = 1.7191 \times 10^{-4} \leq 2^{-12}$.

Da bo napaka $|E_e(g_4)| \leq 2^{-12} - |E_a(g_4)|$, ohranjamo 17 bitov za fiksno vejico pri množenju in koeficiente zaokrožimo na 15 bitov za fiksno vejico. Skupna napaka za g_4 je $|E_a(g_4)+E_e(g_4)| \leq 6.0714\times10^{-5}+5\times2^{-16}+4\times2^{-17} = 1.6753 \times 10^{-4} \leq 2^{-12}$.

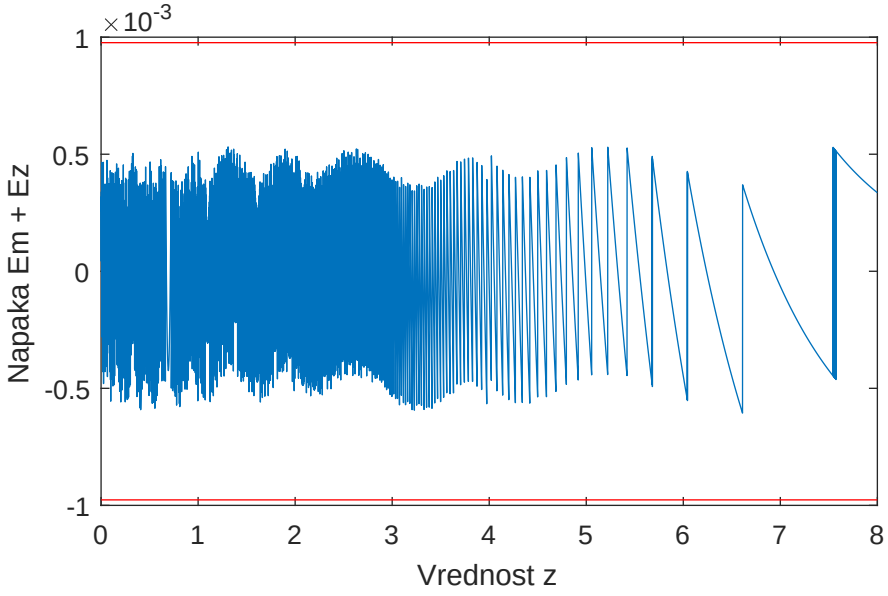
Logaritem $h \ln 2$ zaokrožimo na 15 bitov fiksno vejico. Skupna napaka E_m kompozicije funkcij je $|E_a(g_3) + E_e(g_3) + E_a(g_4) + E_e(g_4) + E_a(h \ln 2)| \leq 3.5470 \times 10^{-4} \leq 2^{-11}$.

Na grafu 7.5 je prikazana skupna napaka E_m. Na grafu 7.6 je prikazana napaka $E_m + E_z$ po končnem zaokroževanju.

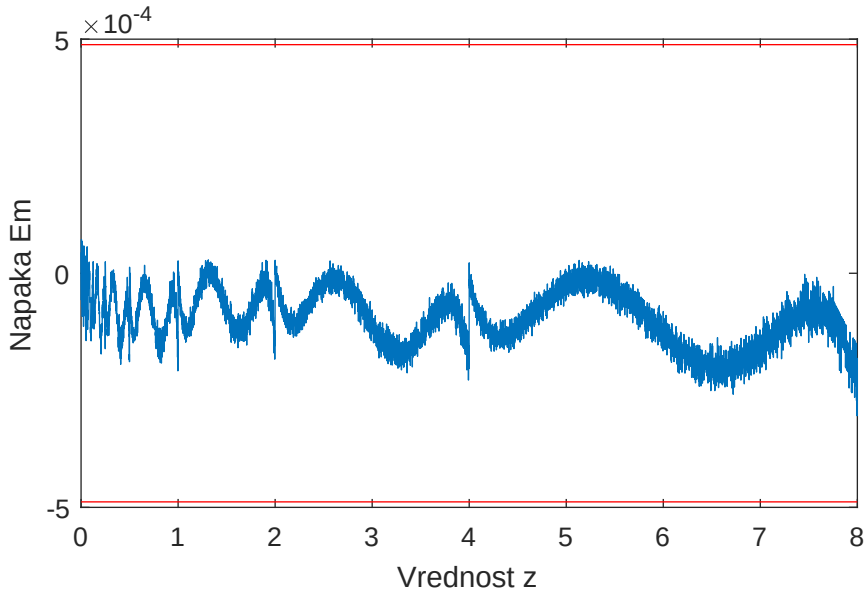
V praksi ohranjanje 15 bitov za fiksno vejico pri množenju nima bistvenega vpliva na končno napako v primerjavi z ohranjanjem 17 bitov za fiksno vejico. Spremembe v napaki se vidijo na grafih 7.7 in 7.8.



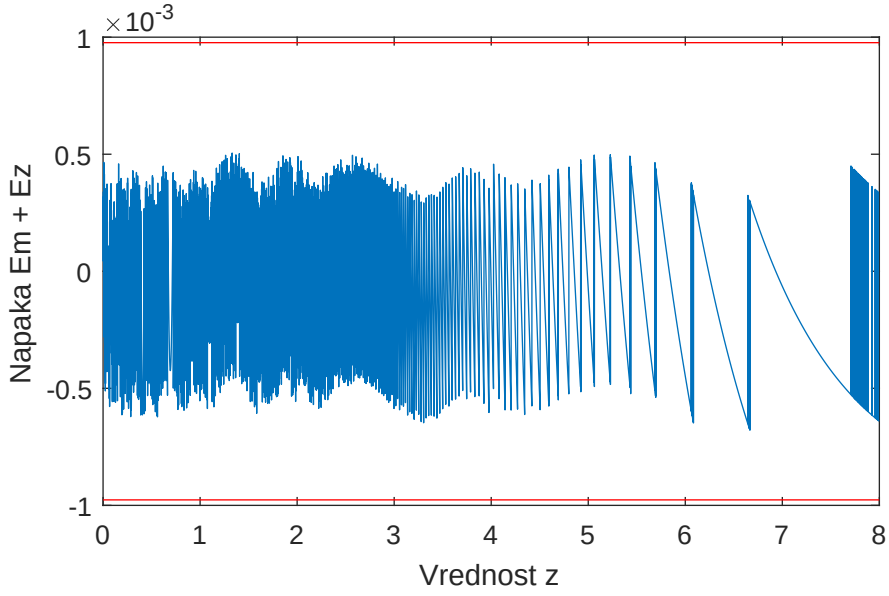
Slika 7.5: Graf napake E_m za g_2 na $[0, 8)$.



Slika 7.6: Graf napake $E_m + E_z$ za g_2 na $[0, 8)$.



Slika 7.7: Graf napake E_m za g_2 z modificiranim množenjem na $[0, 8)$.



Slika 7.8: Graf napake $E_m + E_z$ za g_2 z modificiranim množenjem na $[0, 8)$.

7.2 Enakomerna segmentacija

Evalvacija polinomov visokih stopenj predstavlja problem v logiki (dolgi zakasnilni časi, veliko množilnikov ...). Problem rešujemo s segmentacijo.

7.2.1 Aproksimacija funkcije g_1 v P_1

Interval $[0, 8)$ razdelimo na enako velike podintervale. Vsak podinterval bomo aproksimirali polinomom $p \in P_1$ najboljše enakomerne aproksimacije. Zaradi lažje in boljše implementacije iskanja koeficientov za neki interval v logiki interval razdelimo na podintervale v velikosti $2^i 2^{-10}$.

Zaradi podobnih razlogov in na enak način kot prej uvedemo novo spremenljivko,s čimer skrčimo interval.

Skupna napaka $|E_m|$ ne sme presegati 2^{-11}. Napako aproksimacije in evalvacije uravnoteženo izberemo $|E_a| \leq 2^{-12}$ in $|E_e| \leq 2^{-12}$.

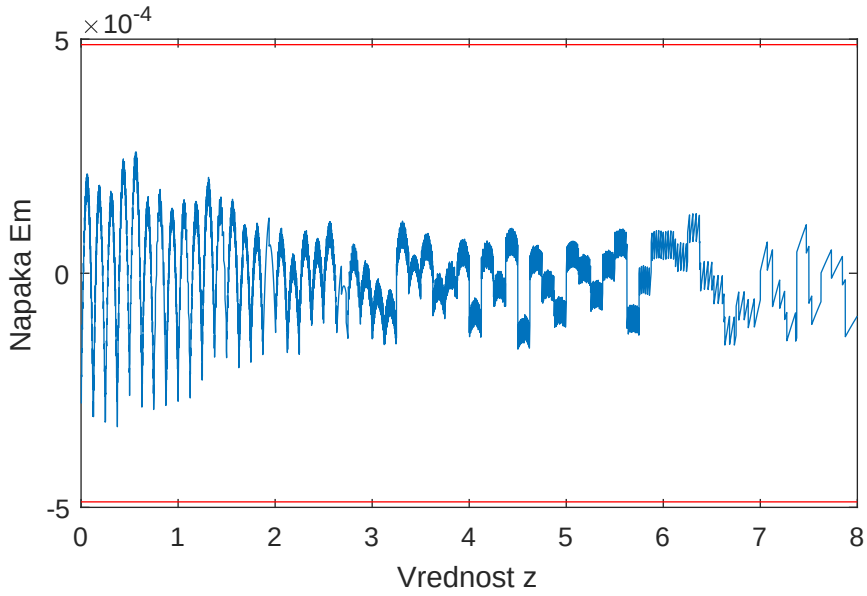
Vodilni koeficient bomo zaokrožili na 12 bitov, začetno vrednost pa na 13 bitov za fiksno vejico. Pri množenju bomo ohranili 14 bitov za fiksno vejico.

Skupna napaka je $|E_m| \leq 2^{-12} + 2^{-13} + 2^{-14} + 2^{-14} \leq 2^{-11}$.

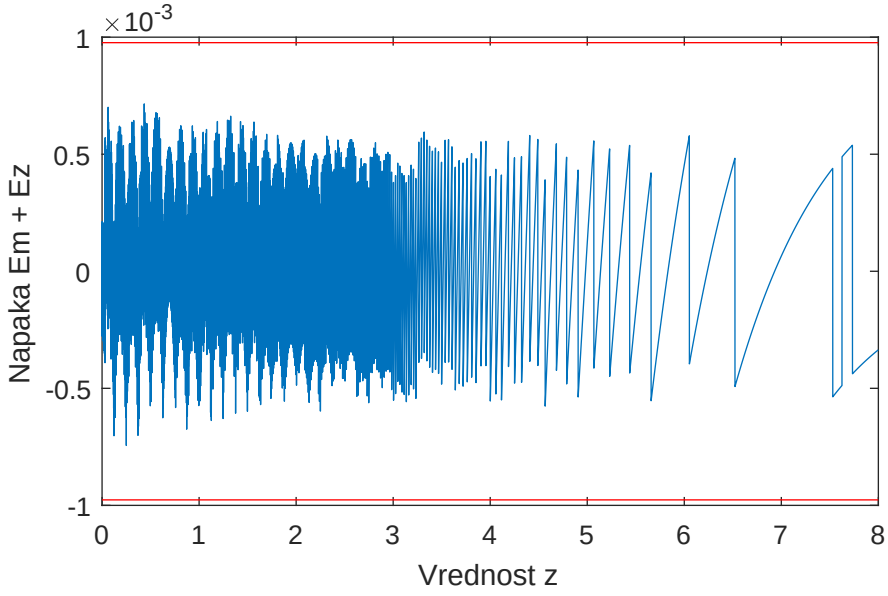
Velikost intervala poiščemo tako, da poskušamo aproksimirati podintervale $[k2^{i-10}, (k+1)2^{i-10}]$, $k = 0, 1, ..., 8 \times 2^{10-i}$ s polinomi v P_1. Če je aproksimacija na vseh podintervalih v mejah napake, smo končali. V nasprotnem primeru zmanjšamo i za ena in postopek ponovimo. Na začetku izberemo $i = 13$.

Da zadostimo potrebam napak, razdelimo interval $[0, 8)$ na 64 enakih delov. Na grafu 7.9 je prikazana napaka E_m. Na grafu 7.10 je prikazana napaka $E_m + E_z$ po končnem zaokroževanju.

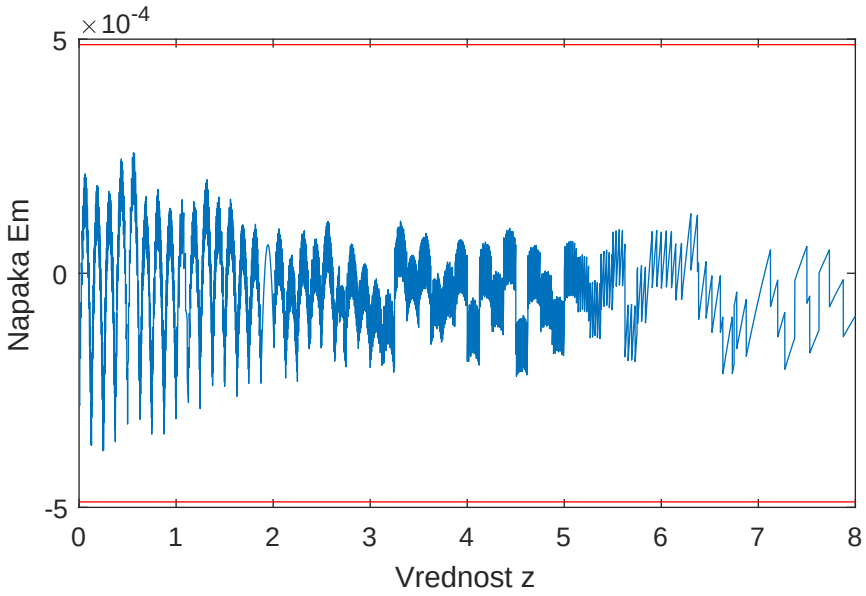
V praksi ohranjanje 13 bitov za fiksno vejico pri množenju nima bistvenega vpliva na končno napako v primerjavi z ohranjanjem 14 bitov za fiksno vejico. Spremembe v napaki se vidijo na grafih 7.11 in 7.12.



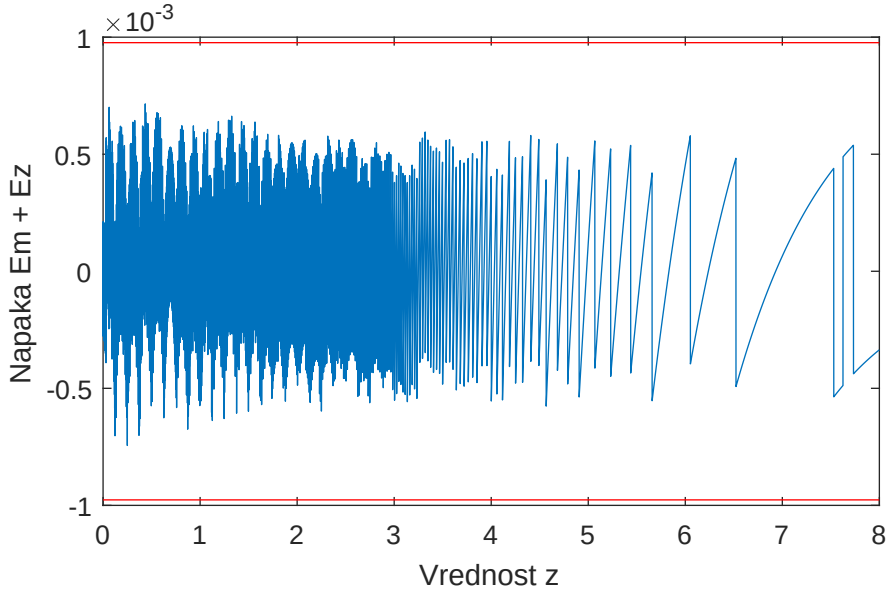
Slika 7.9: Graf napake E_m za g_1 na $[0, 8)$.



Slika 7.10: Graf napake $E_m + E_z$ za g_1 na $[0, 8)$.



Slika 7.11: Graf napake E_m za g_1 z modificiranim množenjem na $[0, 8)$.

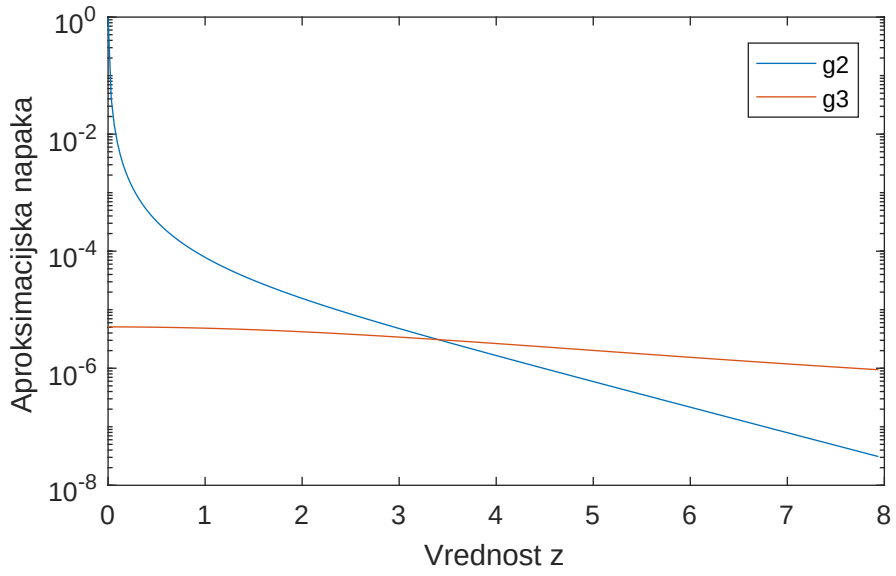


Slika 7.12: Graf napake $E_m + E_z$ za g_1 z modificiranim množenjem na $[0, 8)$.

7.2.2 Aproksimacija funkcije g_2 v P_1

Območje $(0, \gamma)$, kjer bomo g_2 obravnavali kot kompozicijo funkcij, določimo tako, da primerjamo težavnost aproksimacije funkcij g_2 in g_3 s polinomom stopnje ena na neki enakomerni segmentaciji.

Graf 7.13 predstavlja napaki aproksimacije s polinomom stopnje ena pri enakomerni segmentaciji z $\Delta z = 2^{-5}$ funkcij g_2 in g_3.



Slika 7.13: Graf napak pri aproksimaciji s segmentacijo ($\Delta z = 2^{-5}$) funkcij g_2 in g_3 na $[0, 8)$.

Na grafu 7.13 opazimo, da ko je $z \gtrapprox 3.4$ je funkcijo g_2 lažje aproksimirat kot g_3. Zaradi lažje implementacije bomo izbrali $\gamma = 4$ namesto 3.4.

Po uvedbi novih spremenljivk so $g_3(u) = \ln(\frac{e^{4u}-1}{4u})$, $g_4(v) = \ln(2v)$ in $g_2(w) = \ln(e^{8w} - 1)$. Na ta način smo skrčili intervale $(0, 4)$, $[1, 2)$ in $[4, 8)$ na $(0, 1)$, $[0.5, 1)$ in $[0.5, 1)$.

Da zadostimo napakam $|E_e(g_3)| \leq 2^{-13}$, $|E_e(g_4)| \leq 2^{-13}$, vodilne koeficiente pri g_3 in g_4 zaokrožimo na 13 bitov za fiksno vejico, začetne vrednosti pri g_3 in g_4 na 14 bitov za fiksno vejico, pri množenjih pri g_3 in g_4 pa odrežemo bite za 15. bitom za fiksno vejico.

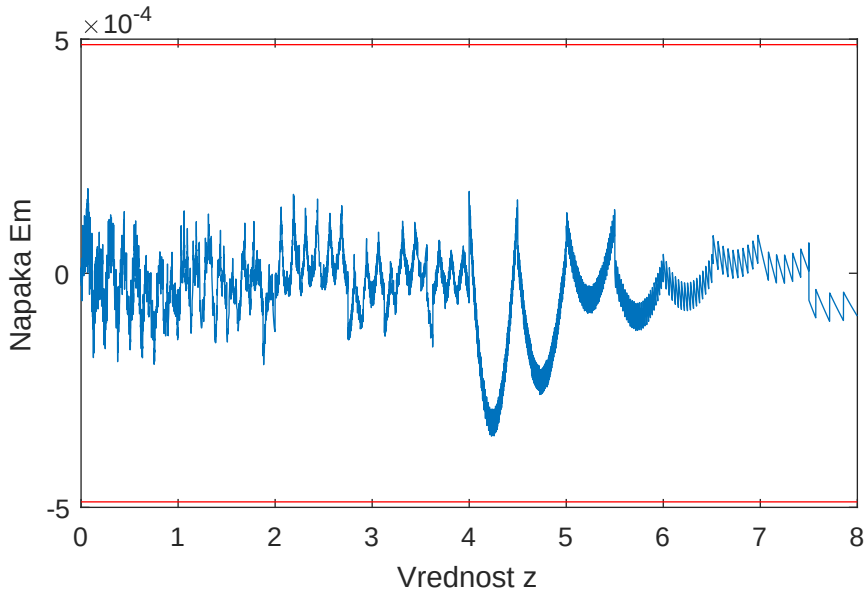
Za izpolnjevanje pogojev napak metode $|E_a(g_3)| \leq 2^{-13}$, $|E_a(g_4)| \leq 2^{-13}$ pri g_3 in g_4, oba intervala razdelimo na 32 enakih delov.

Pri g_2 vodilne koeficiente zaokrožimo na 12 bitov za fiksno vejico, začetne vrednosti zaokrožimo na 13 bitov za fiksno vejico, pri množenjih pa odrežemo bite za 14. bitom za fiksno vejico. Za izpolnjevanje pogojev napak metode interval $[4, 8)$ razdelimo na 8 enakih delov.

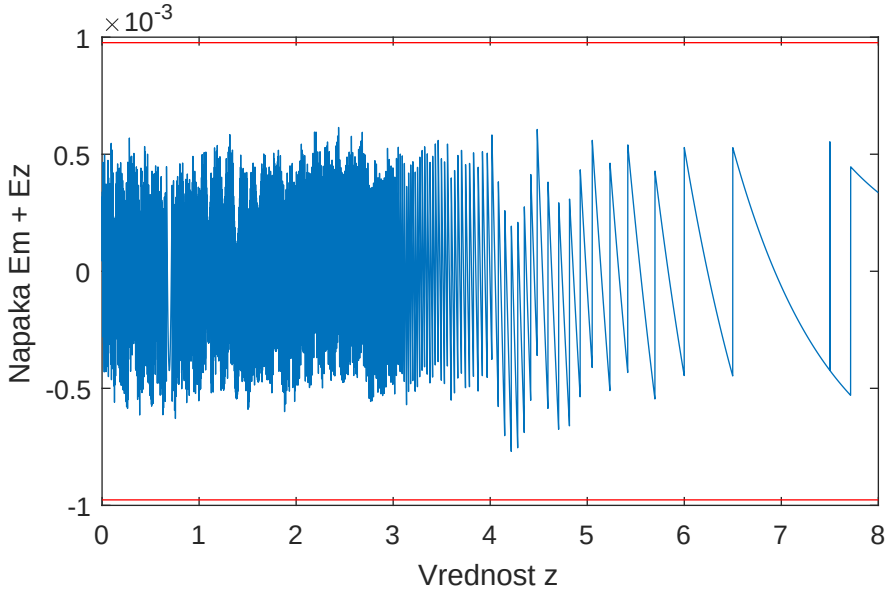
Logaritem $h \ln 2$ zaokrožimo na 13 bitov za fiksno vejico.

Na grafu 7.14 je prikazana napaka E_m. Na grafu 7.15 je prikazana napaka $E_m + E_z$ po končnem zaokroževanju.

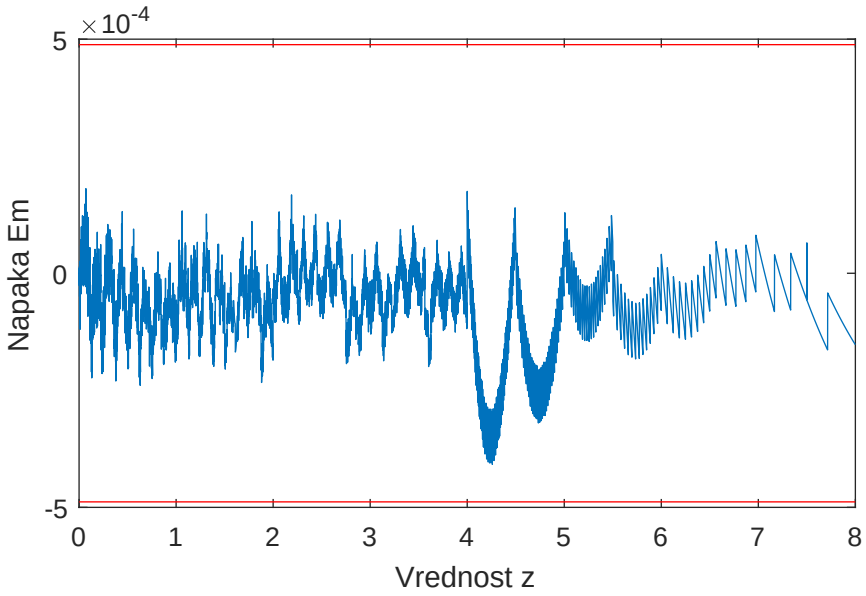
V praksi ohranjanje 13 (g_2) oz. 14 (g_3 in g_4) bitov za fiksno vejico pri množenju nima bistvenega vpliva na končno napako v primerjavi z ohranjanjem 14 oz. 15 bitov za fiksno vejico. Spremembe v napaki se vidijo na grafih 7.16 in 7.17.



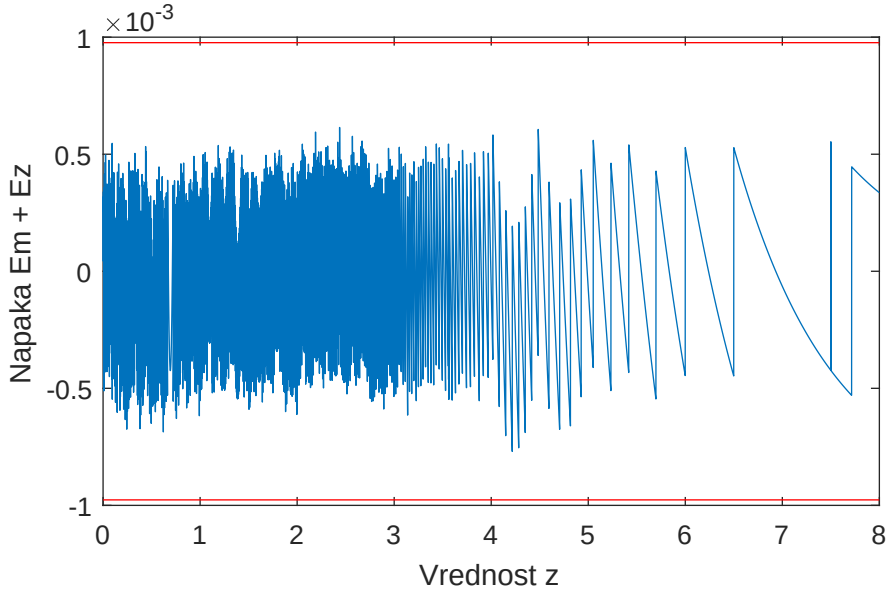
Slika 7.14: Graf napake E_m za g_2 na $[0, 8)$.



Slika 7.15: Graf napake $E_m + E_z$ za g_2 na $[0, 8)$.



Slika 7.16: Graf napake E_m za g_2 z modificiranim množenjem na $[0, 8)$.



Slika 7.17: Graf napake $E_m + E_z$ za g_2 z modificiranim množenjem na $[0, 8)$.

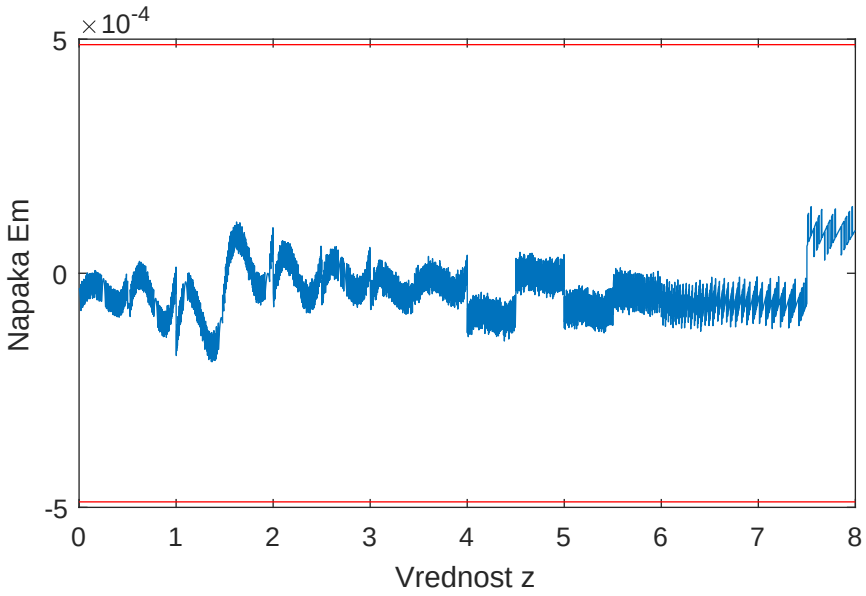
7.2.3 Aproksimacija funkcije g_1 v P_2

Po uvedbi nove spremenljivke je $g_1(u) = \ln(\frac{e^{8u}-1}{8u})$. Na ta način smo skrčili interval z $[0, 8)$ na $[0, 1)$.

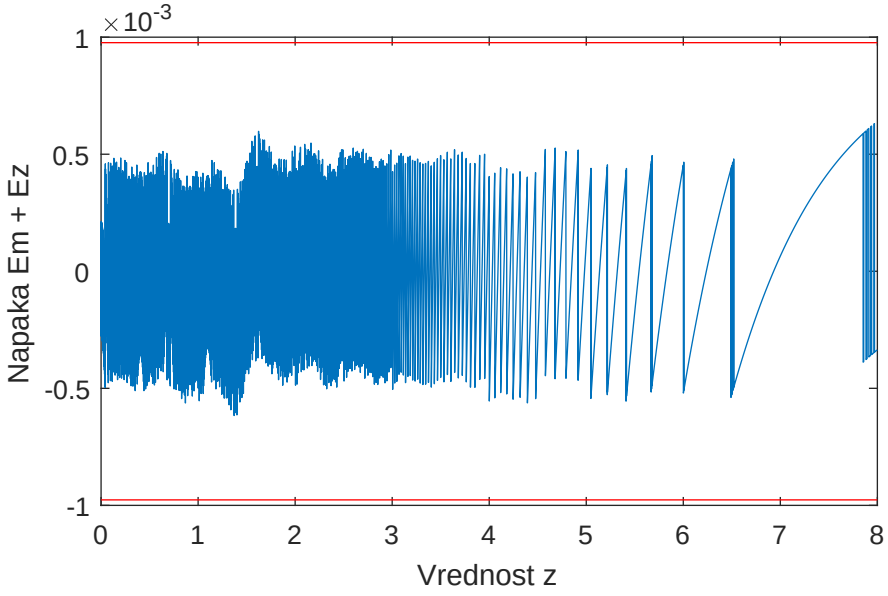
Da zadostimo napaki $|E_e| \leq 2^{-12} - |E_a|$, vse koeficiente zaokrožimo na 13 bitov za fiksno vejico in pri obeh množenjih odrežemo bite za 14. bitom za fiksno vejico.

Za izpolnjevanje pogojev napak metode $|E_a| \leq 2^{-12}$ interval $[0, 8)$ razdelimo na 16 enakih delov. Na grafu 7.18 je prikazana napaka $E_m = E_e + E_a$. Na grafu 7.19 je prikazana napaka $E_m + E_z$ po končnem zaokroževanju.

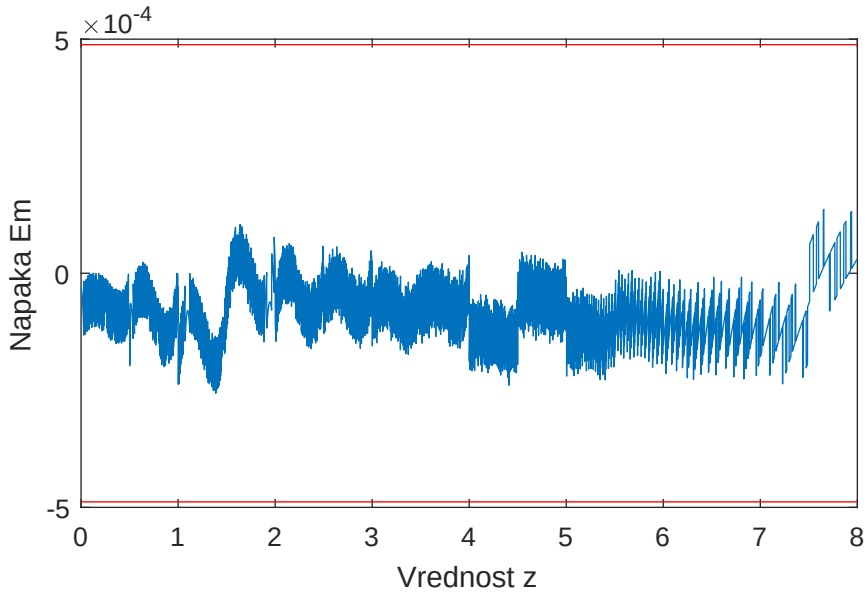
V praksi ohranjanje 13 bitov za fiksno vejico pri množenju nima bistvenega vpliva na končno napako v primerjavi z ohranjanjem 14 bitov za fiksno vejico. Spremembe v napaki se vidijo na grafih 7.20 in 7.21.



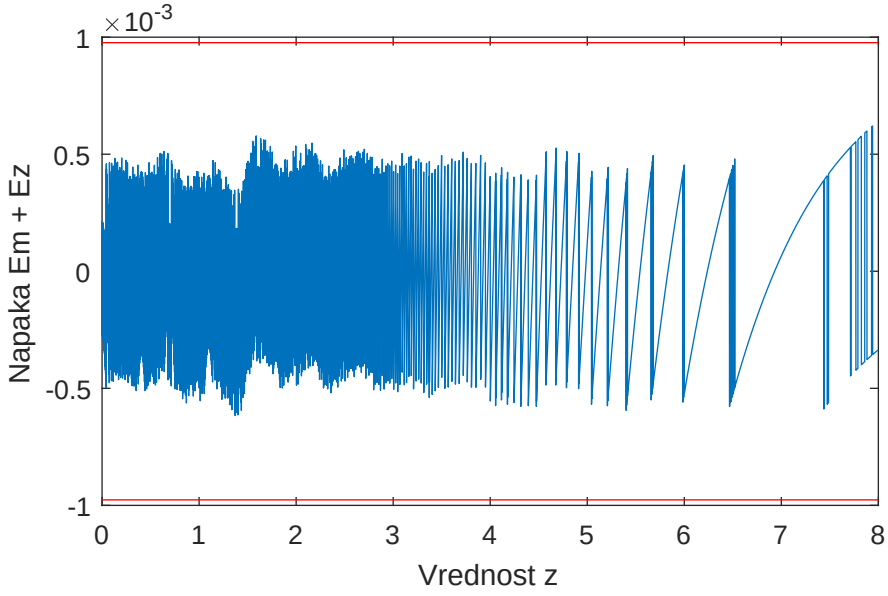
Slika 7.18: Graf napake E_m za $g_1$na $[0, 8)$.



Slika 7.19: Graf napake $E_m + E_z$ za g_1 na $[0, 8)$.



Slika 7.20: Graf napake E_m za g_1 z modificiranim množenjem na $[0, 8)$.

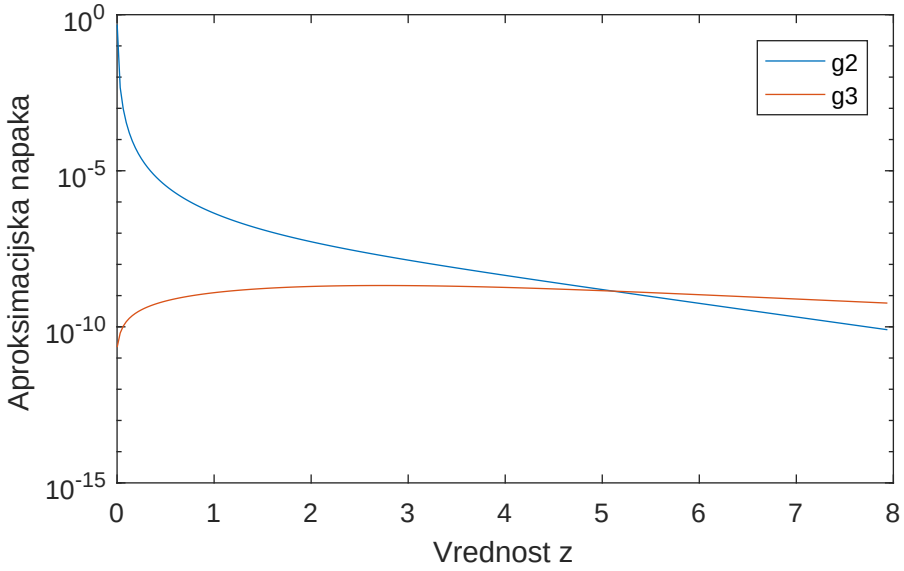


Slika 7.21: Graf napake $E_m + E_z$ za g_1 z modificiranim množenjem na $[0, 8)$.

7.2.4 Aproksimacija funkcije g_2 v P_2

Območje $(0, \gamma)$, kjer bomo g_2 obravnavali kot kompozicijo funkcij, določimo na podoben način kot pri aproksimaciji s polinomi P_1.

Graf 7.22 predstavlja napaki aproksimacije s polinomom stopnje 2 pri enakomerni segmentaciji z $\Delta z = 2^{-5}$ funkcij g_2 in g_3.



Slika 7.22: Graf napak pri aproksimaciji s segmentacijo ($\Delta z = 2^{-5}$) funkcij g_2 in g_3 na $[0, 8)$.

Na grafu 7.22 opazimo, da ko je $z \gtrapprox 5.1$, je funkcijo g_2 lažje aproksimirati kot g_3. Zaradi lažje implementacije bomo izbrali $\gamma = 4$ namesto 5.1.

Uvedemo iste nove spremenljivke kot pri aproksimaciji s polinomi P_1.

Da zadostimo napakam $|E_e(g_3)| \leq 2^{-13}$, $|E_e(g_4)| \leq 2^{-13}$, koeficiente pri g_3 in g_4 zaokrožimo na 14 bitov za fiksno vejico, pri množenjih pri g_3 in g_4 pa odrežemo bite za 16. bitom za fiksno vejico.

Za izpolnjevanje pogojev napak metode $|E_a(g_3)| \leq 2^{-13}$, $|E_a(g_4)| \leq 2^{-13}$ pri g_3 in g_4 oba intervala razdelimo na štiri enake dele.

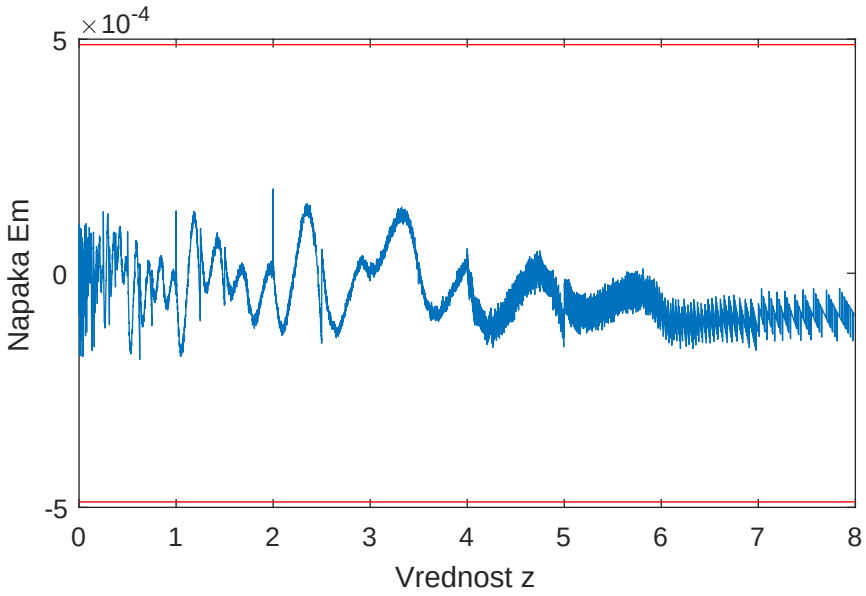
Pri g_2 koeficiente zaokrožimo na 13 bitov za fiksno vejico, pri množenjih pa odrežemo bite za 14. bitom za fiksno vejico. Za izpolnjevanje pogojev

napak metode interval $[4, 8)$ razdelimo na štiri enake dele.

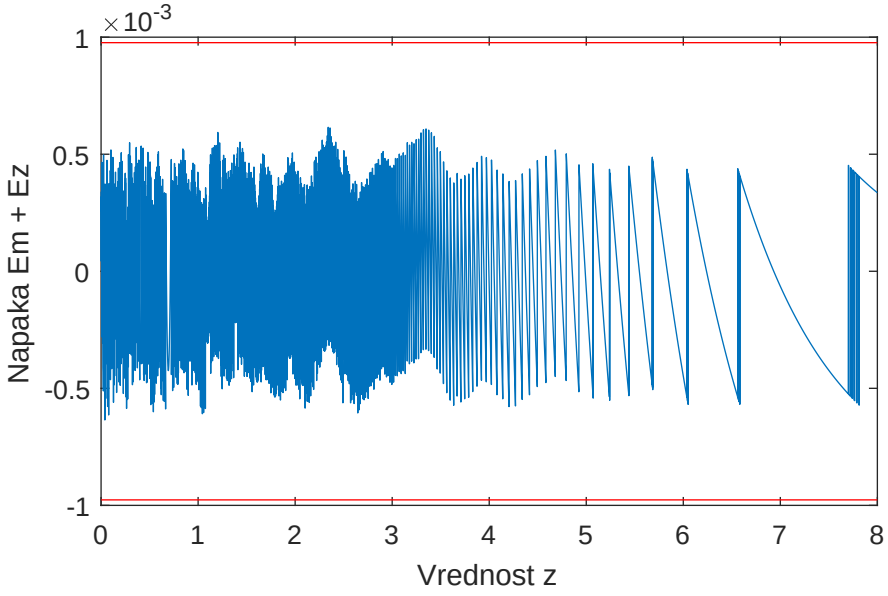
Logaritem $h \ln 2$ zaokrožimo na 14 bitov za fiksno vejico.

Na grafu 7.23 je prikazana napaka $Em = E_e + E_a$. Na grafu 7.24 je prikazana napaka $E_m + E_z$ po končnem zaokroževanju.

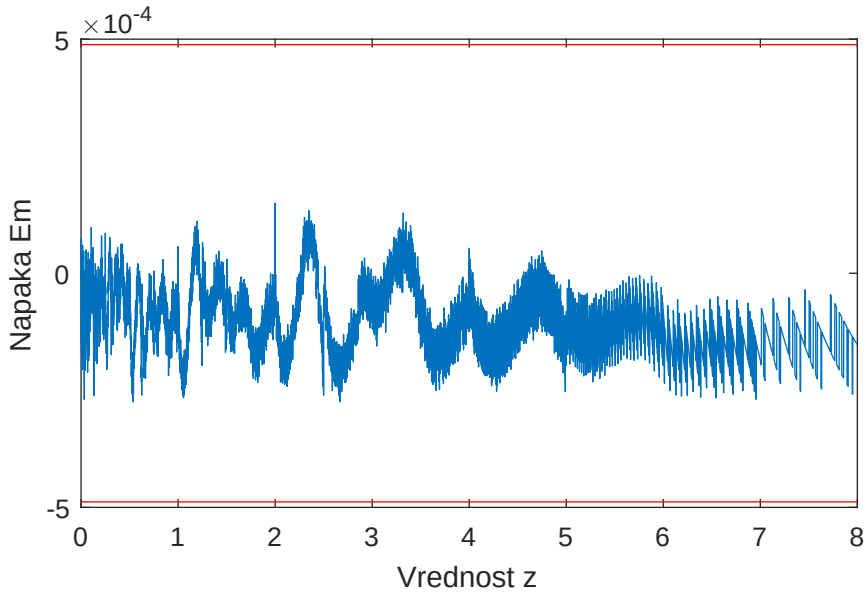
V praksi ohranjanje 13 (g_2) oz. 14 (g_3 in g_4) bitov za fiksno vejico pri množenju nima bistvenega vpliva na končno napako v primerjavi z ohranjanjem 14 oz. 16 bitov za fiksno vejico. Spremembe v napaki se vidijo na grafih 7.25 in 7.26.



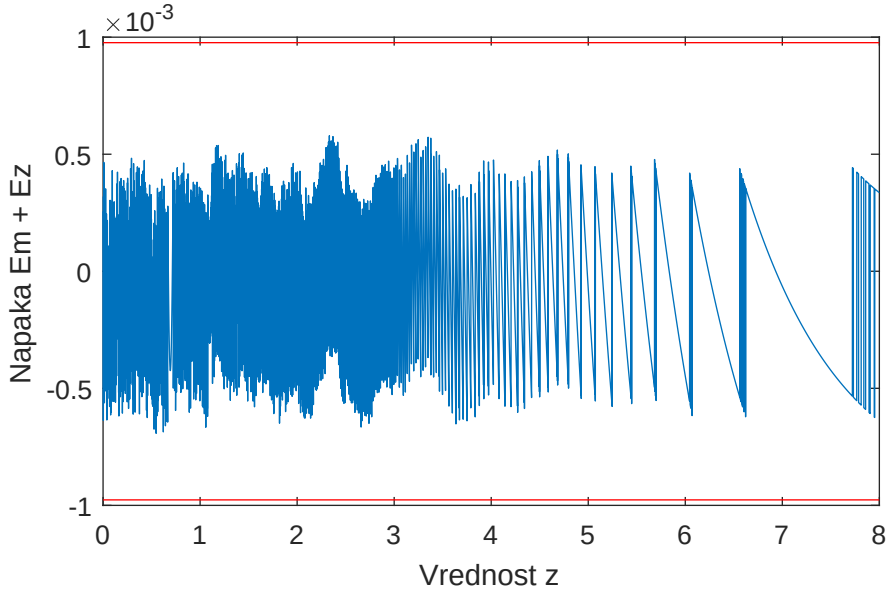
Slika 7.23: Graf napake E_m za g_2 na $[0, 8)$.



Slika 7.24: Graf napake $E_m + E_z$ za g_2 na $[0, 8)$.



Slika 7.25: Graf napake E_m za g_2 z modificiranim množenjem na $[0, 8)$.



Slika 7.26: Graf napake $E_m + E_z$ za g_2 z modificiranim množenjem na $[0, 8)$.

7.3 Neenakomerna segmentacija

Težavnost aproksimacije se na intervalih evalvacije spreminja, zato smo pri enakomerni segmentaciji razdelili intervale na več delov, kot je potrebno, da zadostimo omejitvam aproksimacijske napake. Dele intervala, kjer je aproksimacija funkcije lažja, bomo razdelili na večje podintervale.

Ker je optimalno rešitev (minimizira število podintervalov) računsko težko poiskati, bomo problem reševali s hevristiko. Optimalna rešitev je za implementacijo v logiki nepraktična, ker se porabi veliko logike za iskanje podintervala pri evalvaciji, ker so skoraj vsi podintervali različne velikosti.

Algoritem 2. (Hevristični algoritem za iskanje neenakomerne segmentacije funkcije g na $[a, b]$ v P_n z zgornjo mejo za napako ϵ)

Naj bo s velikost segmenta pri enakomerni segmentaciji za funkcijo g na $[a, b]$. Segmentacijo bomo izvajali od desne proti levi. Z $[q, r]$ označimo interval, ki ga trenutno obravnavamo. Na začetku je $[q, r] = [a, b]$.

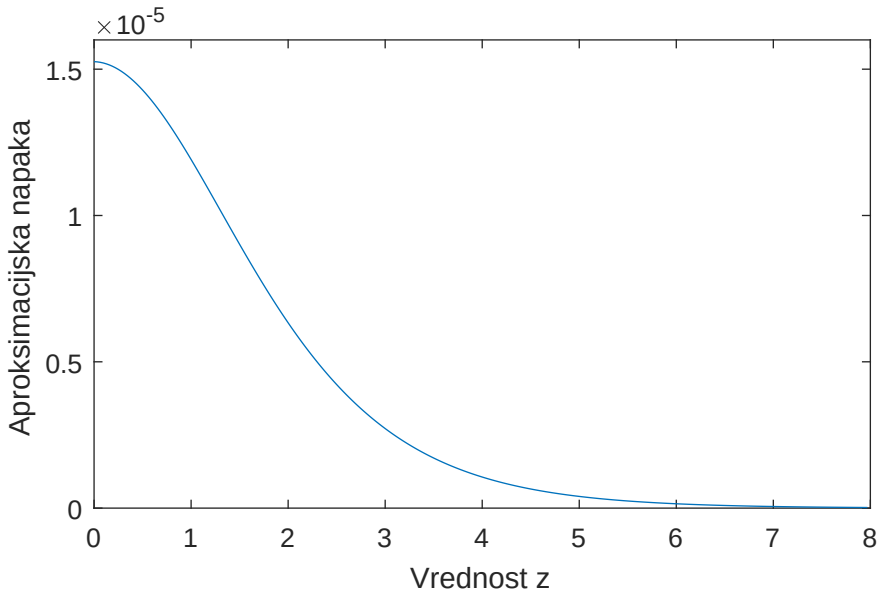
Dokler $[q, r]$ ni prazen interval, ponavljaj:

- inkrementalno poišči maksimalni i, tako da še velja $\min_{p \in P_n} \|g - p\|_{\infty,[r-is,r]} \leq \epsilon$ in $r - is \geq 0$;
- na izhod vrni interval $[r - i\,2^{\lfloor \log_2 s \rfloor}, r]$;
- interval $[q, r]$ zmanjšaj na $[q, r - i\,2^{\lfloor \log_2 s \rfloor}]$.

Če bo težavnost aproksimacije funkcije g v P_n monotono padajoča oz. naraščajoča funkcija, bo algoritem vrnil skoraj optimalno (za implementacijo v logiki) rešitev.

7.3.1 Aproksimacija funkcije g_1 v P_1

Težavnost aproksimacije (napaka aproksimacije v P_1 se izraža v drugem odvodu funkcije g_1) funkcije g_1 na $[0, 8)$ v P_1 je monotono padajoča. To vidimo na grafu 7.27.

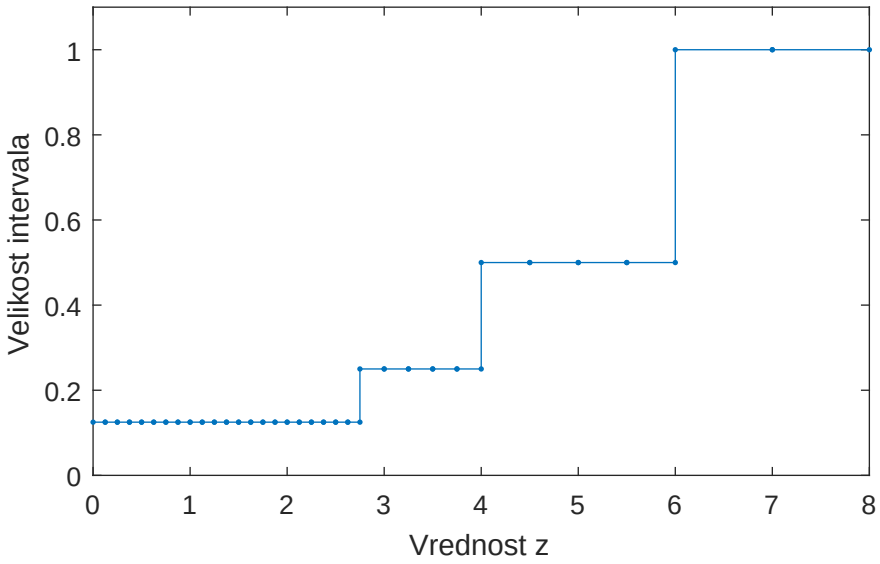


Slika 7.27: Graf napake pri enakomerni aproksimaciji ($\Delta z = 2^{-5}$) na $[0, 8)$.

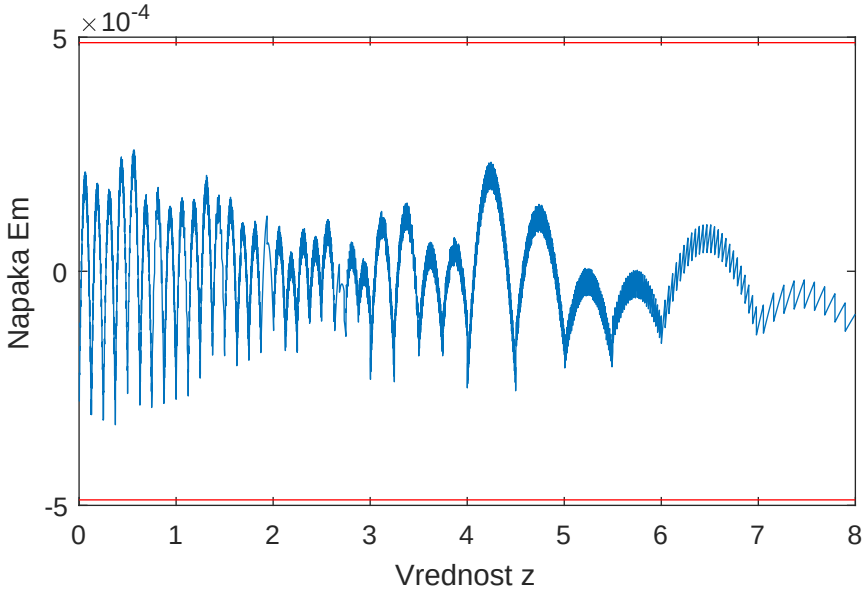
Z upoštevanjem enake napake aproksimacije kot pri enakomerni aproksimaciji v P_1 nam algoritem 2 vrne segmentacijo na 34 delov. Graf 7.28 prikazuje velikost podintervalov v segmentaciji.

Zaradi lažje oz. boljše implementacije v logiki bomo interval $[2.75, 3]$ razdeli na dva enaka podintervala. Zaokroževanje koeficientov in odstranjevanje bitov pri množenju ostanta enaki kot pri enakomerni segmentaciji v P_1. Grafa 7.29 in 7.30 prikazujeta napako pri aproksimaciji, evalvaciji in po končnem zaokroževanju.

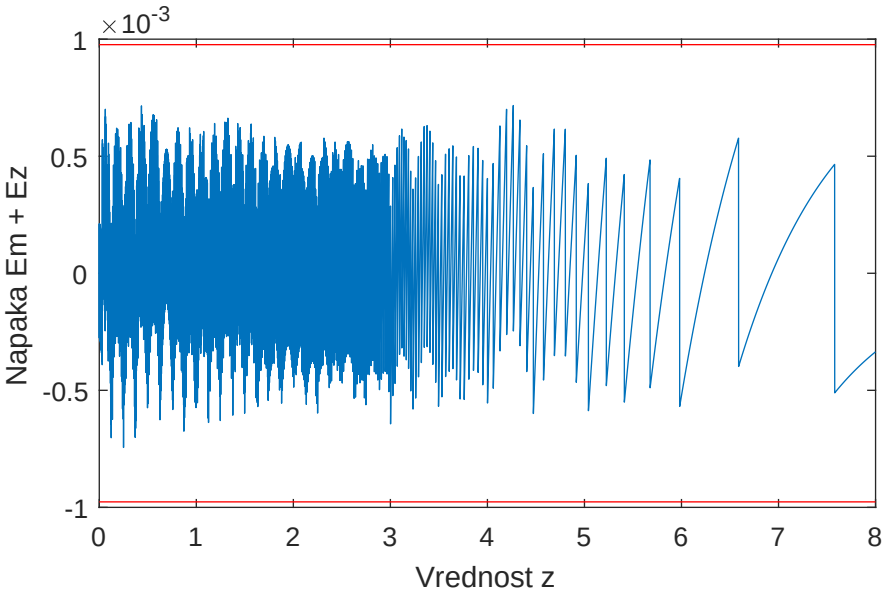
Na enak način kot pri enakomerni segmentaciji lahko modificiramo množenje in zaokroževanje koeficientov, da dodatna napaka nima bistvenega vpliva. Spremembo napake opazimo na grafih 7.31 in 7.32.



Slika 7.28: Segmetacija funkcije g_1 v P_1 na $[0, 8)$.

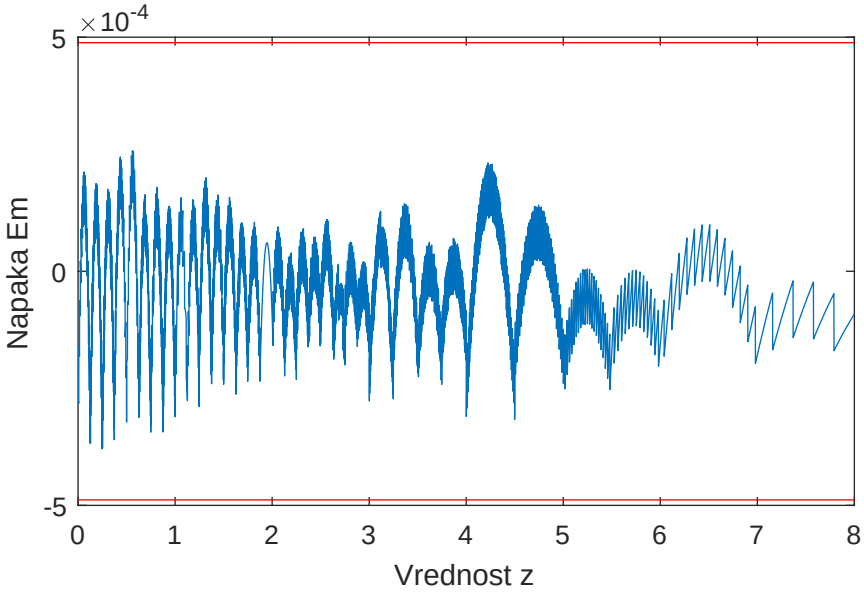


Slika 7.29: Graf napake E_m za g_1 na $[0, 8)$.

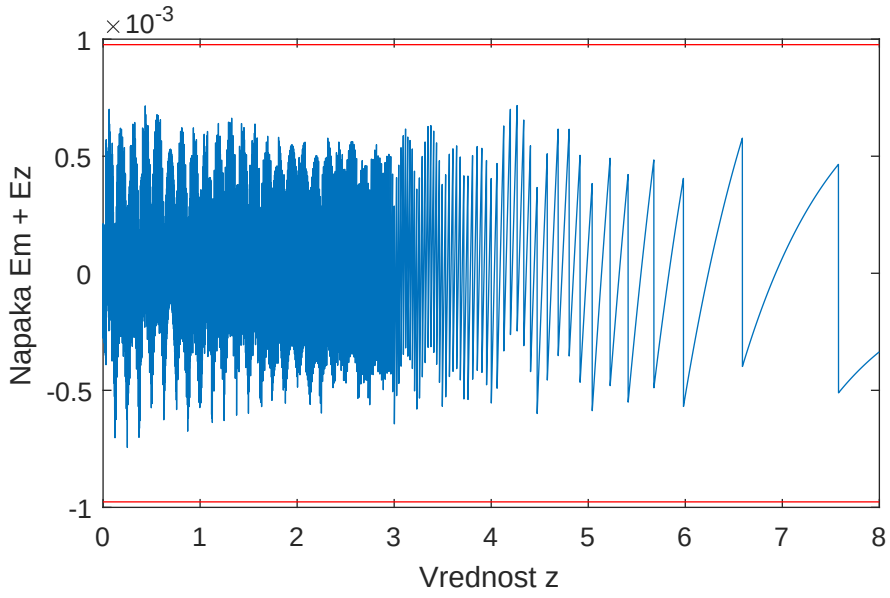


-

Slika 7.30: Graf napake $E_m + E_z$ za g_1 na $[0, 8)$.



Slika 7.31: Graf napake E_m za g_1 z modificiranim množenjem na $[0, 8)$.

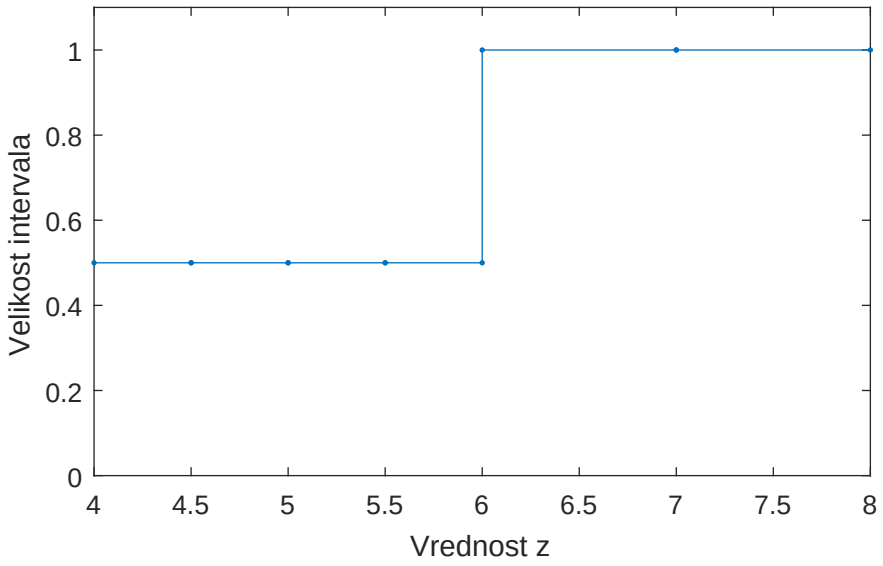


Slika 7.32: Graf napake $E_m + E_z$ za g_1 z modificiranim množenjem na $[0, 8)$.

7.3.2 Aproksimacija funkcije g_2 v P_1

Pri g_2 bomo funkcijo razdeli na funkcije g_2, g_3 in g_4, in sicer na enak način kot pri enakomerni segmentaciji g_2 v P_1. Meje za napake pri aproksimaciji, zaokroževanje koeficientov in množenje ostanejo enaki.

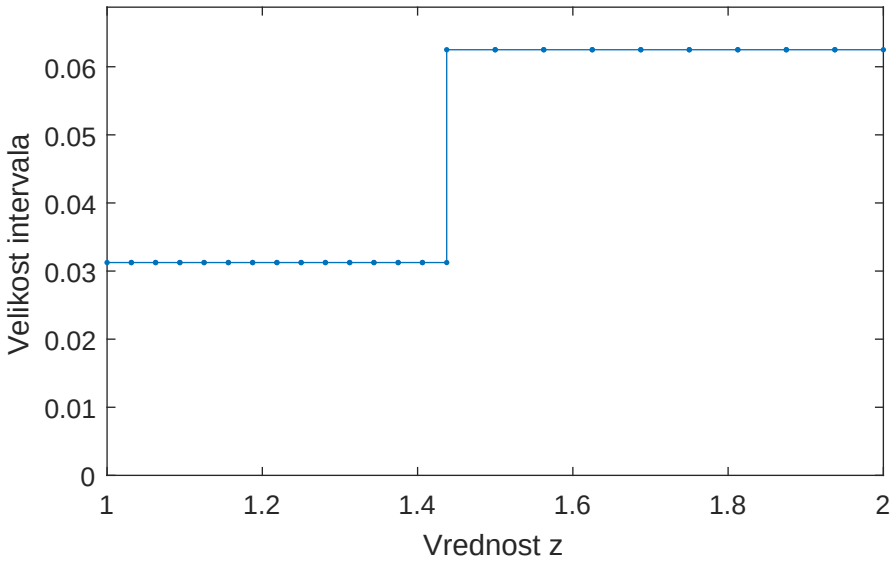
Pri g_3 nam algoritem 2 vrne enakomerno segmentacijo na 32 delov. Za funkciji g_2 in g_4 dobimo segmentaciji na 6 in 23 delov. Grafa 7.33 in 7.34 prikazujeta velikost podintervalov v segmentaciji funkcije g_2 in g_4.



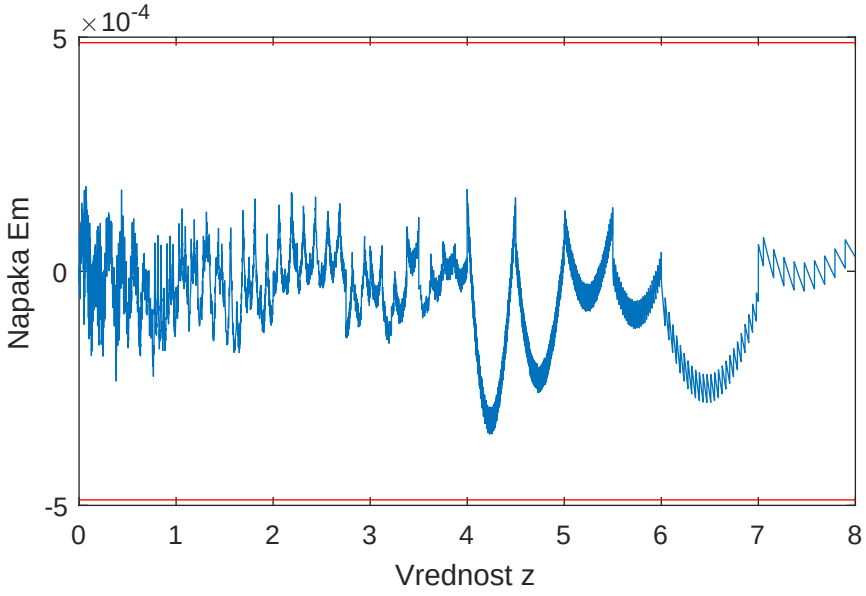
Slika 7.33: Segmetacija funkcije g_2 v P_1 na $[4, 8)$.

Zaradi lažje oz. boljše implementacije v logiki bomo interval $[1.4375, 1.5]$ razdeli na dva enaka podintervala. Grafa 7.35 in 7.36 prikazujeta napako pri aproksimaciji, evalvaciji in po končnem zaokroževanju.

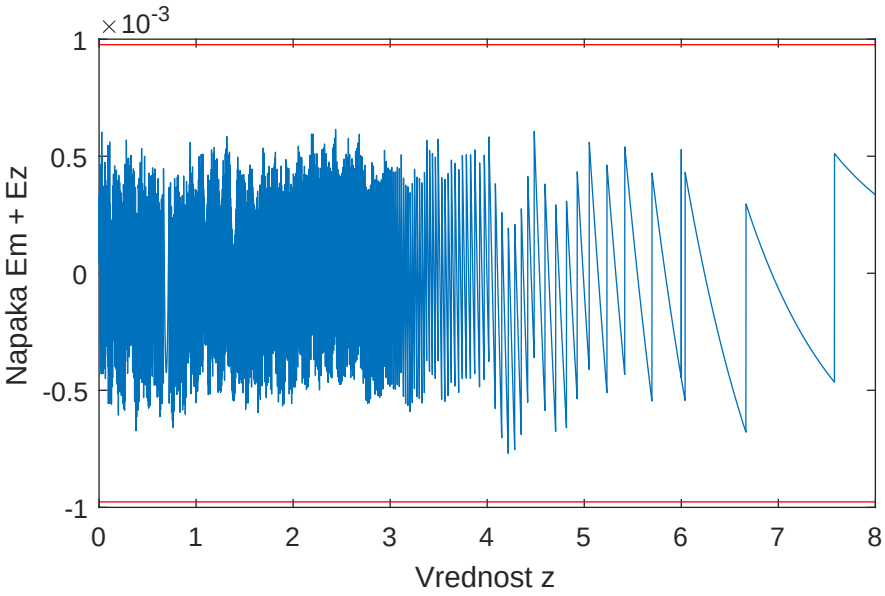
Na enak način kot pri enakomerni segmentaciji lahko modificiramo množenje in zaokroževanje koeficientov, da dodatna napaka nima bistvenega vpliva. Spremembo napake opazimo na grafih 7.37 in 7.38.



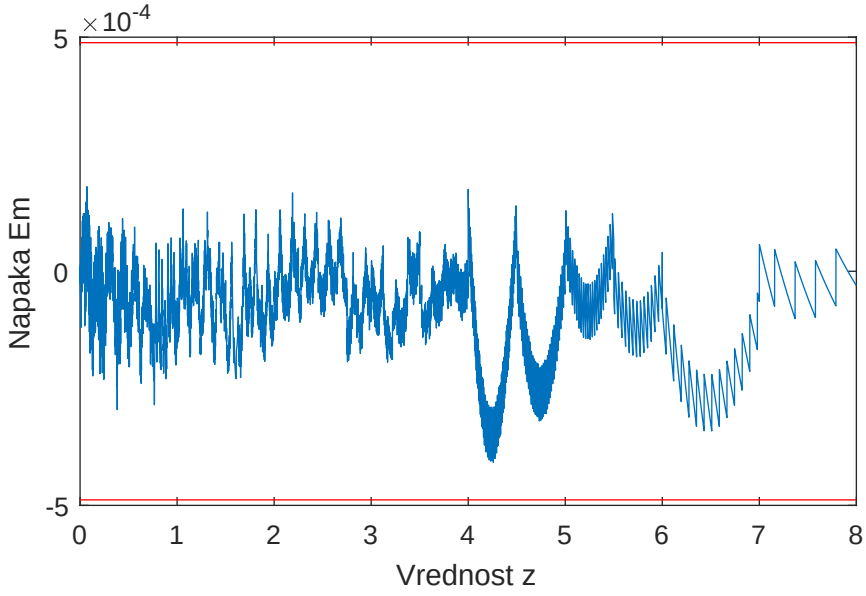
Slika 7.34: Segmetacija funkcije g_4 v P_1 na $[1, 2)$.



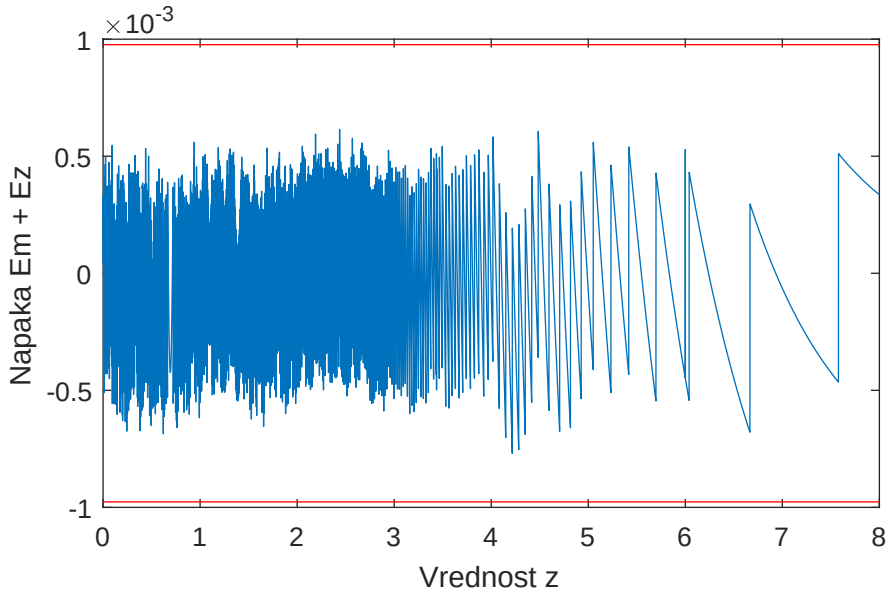
Slika 7.35: Graf napake E_m za g_2 na $[0, 8)$.



Slika 7.36: Graf napake $E_m + E_z$ za g_2 na $[0, 8)$.



Slika 7.37: Graf napake E_m za g_2 z modificiranim množenjem na $[0, 8)$.

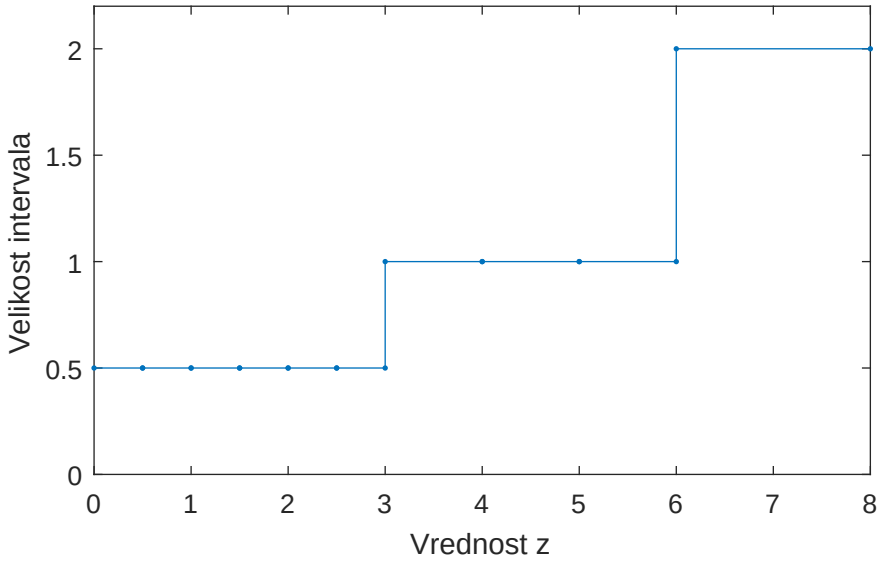


Slika 7.38: Graf napake $E_m + E_z$ za g_2 z modificiranim množenjem na $[0, 8)$.

7.3.3 Aproksimacija funkcije g_1 v P_2

Meje za napake pri aproksimaciji, zaokroževanje koeficientov in množenje ostanjo enaki kot pri enakomerni aproksimaciji funkcije g_1 v P_2.

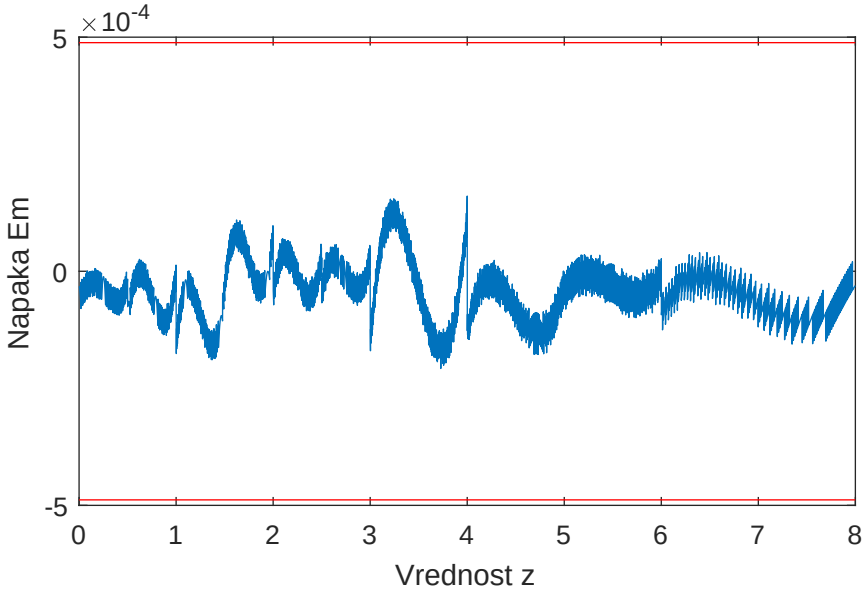
Za funkcijo g_1 v P_2 nam algoritem 2 vrne segmentacijo na 10 delov. Graf 7.39 prikazuje velikost podintervalov v segmentaciji.



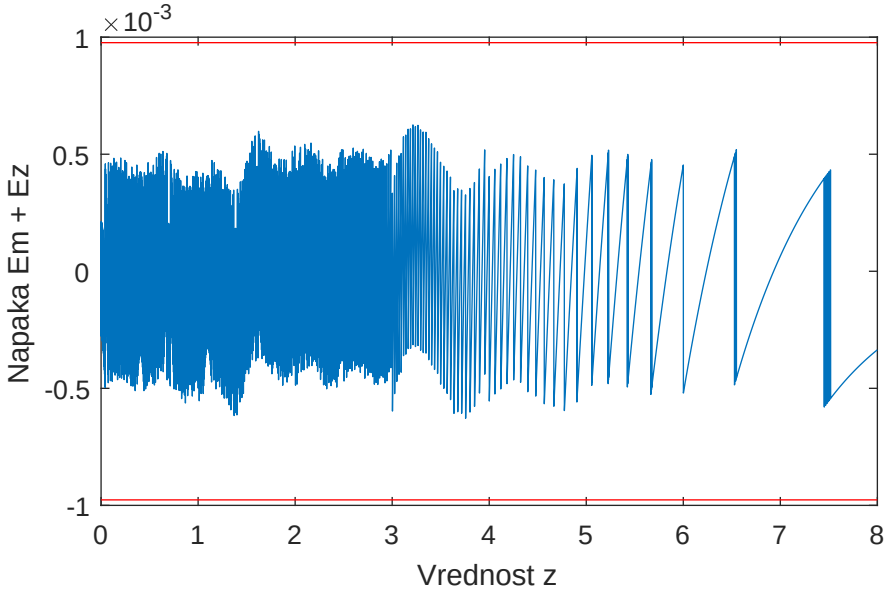
Slika 7.39: Segmetacija funkcije g_1 v P_2 na $[0, 8)$.

Grafa 7.40 in 7.41 prikazujeta napako pri aproksimaciji, evalvaciji in po končnem zaokroževanju.

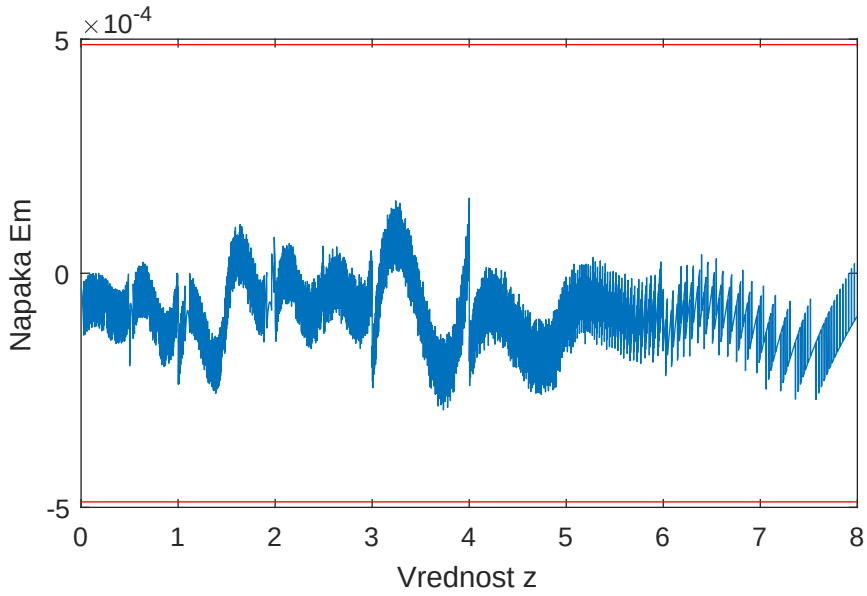
Na enak način kot pri enakomerni segmentaciji lahko modificiramo množenje in zaokroževanje koeficientov, da dodatna napaka nima bistvenega vpliva. Spremembo napake opazimo na grafih 7.42 in 7.43.



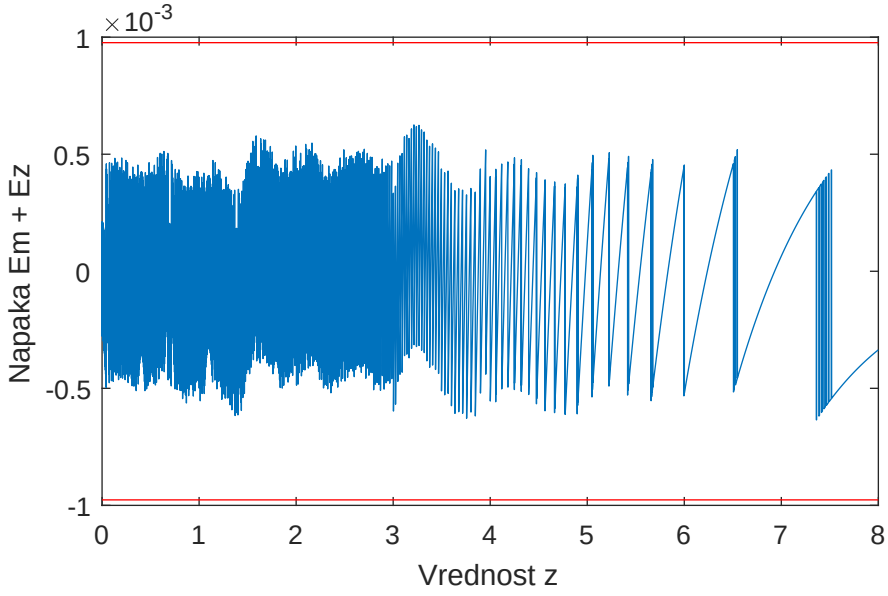
Slika 7.40: Graf napake E_m za g_1 na $[0, 8)$.



Slika 7.41: Graf napake $E_m + E_z$ za g_1 na $[0, 8)$.



Slika 7.42: Graf napake E_m za g_1 z modificiranim množenjem na $[0, 8)$.

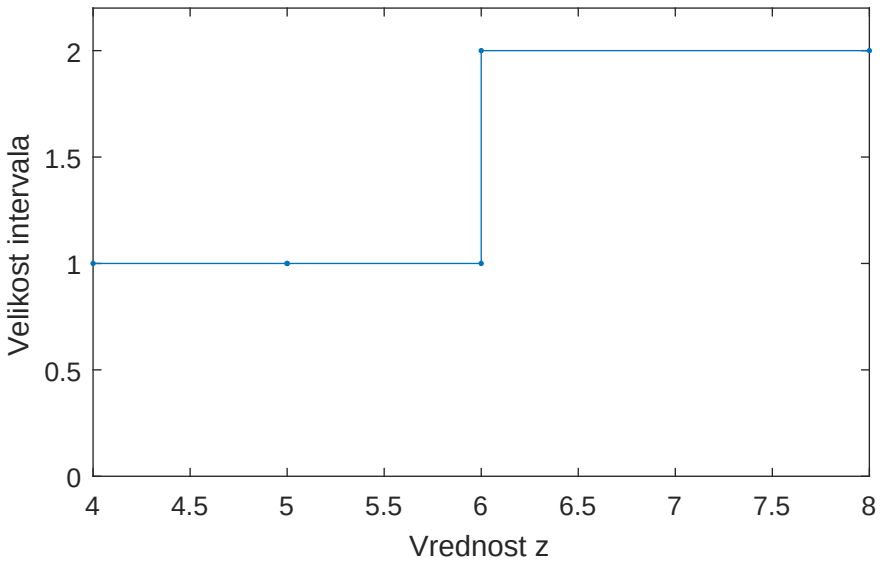


Slika 7.43: Graf napake $E_m + E_z$ za g_1 z modificiranim množenjem na $[0, 8)$.

7.3.4 Aproksimacija funkcije g_2 v P_2

Pri g_2 bomo funkcijo razdeli na funkcije g_2, g_3 in g_4, in sicer na enak način kot pri enakomerni segmentaciji g_2 v P_2. Meje za napake pri aproksimaciji, zaokroževanje koeficientov in množenje ostanejo enaki.

Pri g_3 in g_4 nam algoritem 2 vrne enakomerno segmentacijo na štiri dele. Za funkcijo g_2 dobimo segmentacijo na tri dele. Graf 7.44 prikazuje velikost podintervalov v segmentaciji.



Slika 7.44: Segmetacija funkcije g_2 v P_1 na $[4, 8)$.

Edina sprememba v primerjavi z enakomerno segmentacijo je združitev zadnjih dveh podintervalov v g_2, zato se tudi napake malo razlikujejo od napak pri analogu z enakomerno segmentacijo.

7.4 Opombe in opažanja

V naši implementaciji je skupna napaka bistveno manjša od omejitve 2^{-10}. Določanje načina aproksimacije, zaokroževanja in množenja tako, da smo zelo blizu mejam napak, ni preprosto, ker imamo veliko odvisnih nezveznih

parametrov. Pri izberi parametrov moramo paziti, da je implementacija v logiki smiselna in da je približno polovica napake v pozitivni, polovica v pa negativni smeri. Če imamo balansirano napako glede na pozitivnost oz. negativnost napake, lahko akumulacijo napake v določenih primerih zelo omejimo. Tak primer je Kahanov sumacijski algoritem [5], ki korensko $\mathcal{O}(\sqrt{n})$ rast napake pri zaporednem seštevanju n-tih števil v premični vejici omeji na konstantno rast $\mathcal{O}(1)$. V splošnem je pri balansirani napaki rast napake pri n zaporednih operacijah korenska $\mathcal{O}(\sqrt{n})$ (random walk) [14, 9], v primeru da napaka ni balansirana, pa linearna $\mathcal{O}(n)$.

Končna napaka pri seštevanju in odštevanju v 16-bitnem LNS je enaka napakam aproksimacije in evalvacije ter končnemu zaokroževanju funkcij g_1 in g_2, če je logaritemska absolutna razlika med vhodnima številoma manjša od 8. V nasprotnem primeru je napaka manjša od 2^{-11}.

Tabela 7.1 vsebuje za posamezno metodo število koeficientov polinomov. Z neenakomerno segmentacijo smo pri P_1 zmanjšali število koeficientov za približno 28 %, pri P_2 pa za približno 22 %.

Tabela 7.2 vsebuje povprečno napako za posamezno metodo. Tabela 7.3 vsebuje podatek od deležu napak, ki so manjše od 2^{-11} (točno zaokroževanje).

metoda	# koeficientov
v P_n	$8+7+5+12=32$
v P_1 enak. s.	$(64+32+32+8)\times 2+12=284$
v P_2 enak. s.	$(16+4+4+4)\times 3+12=96$
v P_1 neenak. s.	$(34+32+24+6)\times 2+12=204$
v P_2 neenak. s.	$(10+4+4+3)\times 3+12=75$

Tabela 7.1: Število koeficientov pri posamezni metodi.

metoda	povp. $\lvert E(g_1)\rvert$	povp. $\lvert E(g_2)\rvert$
v P_n	2.9768×10^{-4}	2.7012×10^{-4}
v P_1 enak. s.	2.6116×10^{-4}	2.6150×10^{-4}
v P_2 enak. s.	2.6000×10^{-4}	2.6402×10^{-4}
v P_1 neenak. s.	2.6259×10^{-4}	2.6510×10^{-4}
v P_2 neenak. s.	2.5801×10^{-4}	2.7103×10^{-4}

Tabela 7.2: Primerjava napak pri posamezni metodi.

metoda	delež n. $\lvert E(g_1)\rvert\leq 2^{-11}$	delež n. $\lvert E(g_2)\rvert\leq 2^{-11}$
v P_n	0.7969	0.8850
v P_1 enak. s.	0.9164	0.9297
v P_2 enak. s.	0.9341	0.9095
v P_1 neenak. s.	0.9210	0.9165
v P_2 neenak. s.	0.9447	0.8794

Tabela 7.3: Delež napak $\lvert E\rvert\leq 2^{-11}$.

Poglavje 8

Implementacija – FPGA

Pri digitalnem načrtovanju aritmetične enote v 16-bitnem LNS smo uporabili Digilent-tov Nexys 2 z Xilingx-ovim Spratan3E-500 FPGA-jem [7]. Logiko smo načrtovali v jeziku VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) v razvojnem okolju Xilinx ISE Design Suite.

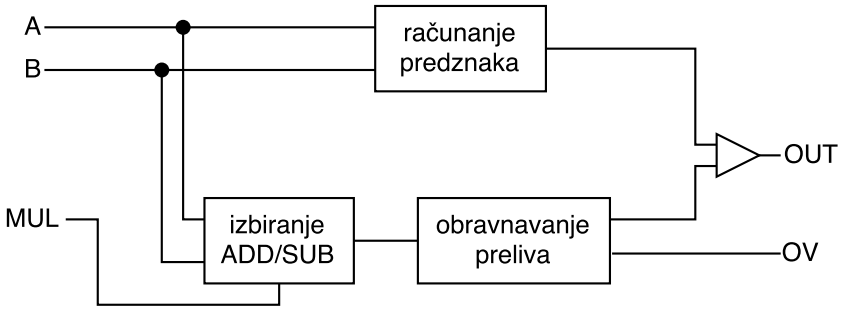
8.1 Implementacija aritmetične enote v 16-bitnem LNS

Enota vsebuje množenje, deljenje, seštevanje in odštevanje. Množenje in deljenje sta v enoti ločeni od seštevanja in odštevanja (neodvisnost). Implementacija aritmetične enote je asinhronska. Funkcije g_1, g_2, g_3 in g_4 v enoti za seštevanje in odštevanje bomo aproksimirali ter evalvirali v P_1 z neenakomerno segmentacijo.

8.1.1 Enota za množenje in deljenje

Vhod enote sta dve 16-bitni LNS-števili A in B ter vrsta operacije MUL (množenje oz. deljenje). Izhod OUT je zmnožek oz. količnik in indikator preliva OV. Skica enote je na sliki 8.1.

Glede na MUL izberemo seštevanje (MUL = 1) oz. odštevaje (MUL =



Slika 8.1: Skica enote za množenje in deljenje.

0) logaritmov števil A in B. Pri seštevanju oz. odštevanju lahko pride do preliva. Do preliva pride, ko sta oba logaritma pozitivna oz. negativna, rezultat pa negativen oz. pozitiven. V takih primerih se rezultat zaokroži na prekoračitev oz. podkoračitev. Predznak izhoda OUT je XOR-operacija nad predznakoma A in B. Če implementiramo pravilno množenje z LNS(0), so potrebni trije dodatni 15-bitni primerjalniki nad logaritmoma števil A in B.

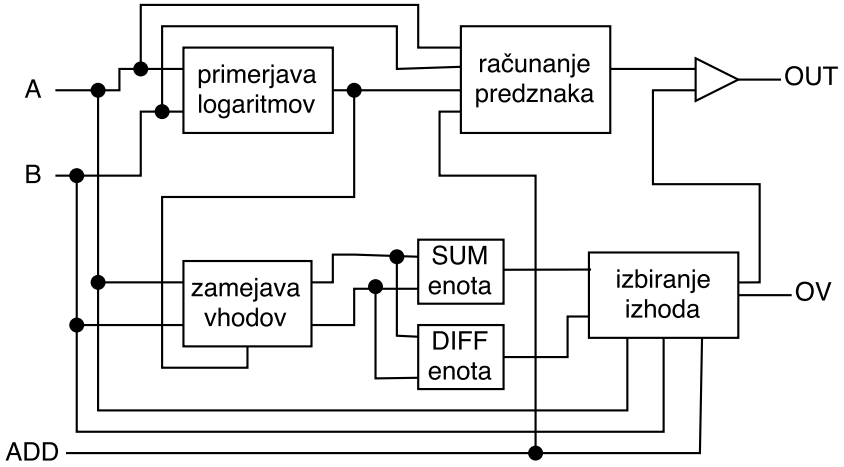
8.1.2 Enota za seštevanje in odštevanje

Vhod enote sta dve 16-bitni LNS-števili A in B ter vrsta operacije ADD (seštevanje oz. odštevanje). Izhod OUT je vsota oz. razlika in indikator preliva OV. Skica enote je na sliki 8.2.

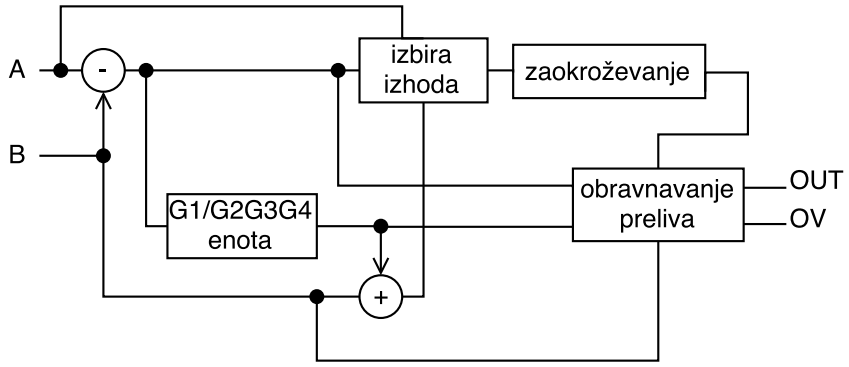
Vhoda zamenjamo, če je logaritem pri A manjši kot pri B. Predznak izhoda in izbiranje izhoda diferenčne ter sumacijske enote izberemo na način, ki je opisan v tabeli v poglavju 4.3. Diferenčna in sumacijska enota vračata tudi indikator preliva. Splošna skica diferenčne oz. sumacijske enote je predstavljena na sliki 8.3.

Glede na to, ali imamo sumacijsko oz. diferenčno enoto, različno obravnavamo bloke na shemi s sliko 8.3.

- Sumacijska enota:
  - Enota G1/G2G3G4: enota G1 z vhodom razlike logaritmov A in



Slika 8.2: Skica enote za množenje in deljenje.



Slika 8.3: Skica diferenčne oz. sumacijske enote.

B.

– Izbiranje izhoda: če je razlika med logaritmoma A in B manjša od 8 (dvobitni primerjalnik), izberemo vsoto logaritma B in izhoda enote G1, sicer izberemo A.
– Zaokroževanje: zaokrožujemo na najbližje predstavljivo število. Če je število na sredini, zaokrožimo tako, da bo na zadnjem mestu nič. Zaokroževanje ne vpliva na preliv.
– Obravnavanje preliva: če je razlika med logaritmoma A in B manjša od 8, logaritem B pozitiven in vsota logaritma B ter iz-

hoda enote G1 negativna, potem je prišlo do preliva OV=1 in je izhod LNS(∞).

- Diferenčna enota:
  - Enota G1/G2G3G4: enota G2G3G4 z vhodom razlike logaritmov A in B.
  - Izbiranje izhoda: enako kot pri sumacijski enoti.
  - Zaokroževanje: ker je logaritem prestavljen v dvojiškem komplementu, zaokrožujemo negativna števila na popolnoma enak način kot pozitivna. Zato je zaokroževanje enako kot pri sumacijski enoti.
  - Obravnavanje preliva: če je razlika med logaritmoma A in B manjša od 8, logaritem B negativen, izhod enote G2G3G4 negativen in vsota logaritma B ter izhoda enote G2G3G4 pozitivna, potem je prišlo do preliva OV=1 in je izhod LNS(0). Če je razlika med logaritmoma A in B enaka nič, potem je prišlo do preliva in OV=1 in je izhod LNS(0).

8.1.3 Enota G1

Vhod Z je 13-bitni (fiksna vejica b3.10). Ker segmentacija razdeli interval na 34 delov, bomo zadnja dva intervala ($[6,7)$, $[7,8)$), implementirali z logiko, ostali koeficienti 32 intervalov pa bodo hranjeni v 32×26 velikem ROM-u. V ROM-u hranimo oba koeficienta polinoma $p \in P_1$ zaporedoma na istem naslovu. Prvih 14 bitov (implicitno 15) za vodilni koeficient in zadnjih 13 bitov za začetno vrednost. Ker je prvi bit pri vodilnem koeficientu vedno ena, hranimo ta bit posebej. Ker imamo bločno enakomerno segmentacijo, ne moremo direktno uporabiti vhoda Z za naslov intervala v ROM-u. Naslov izračunamo tako, da s tremi 3-bitnimi primerjalniki prvih treh bitov vhoda Z preverimo, na katerem od štirih blokov ($[0,3)$,$[3,4)$,$[5,6)$ in $[6,8)$) smo. Za lokacijo v bloku uporabimo direktno bite iz vhoda Z. Izkaže se, da za računaje

naslova ni treba seštevati pomika (glede na blok) in naslova iz vhoda Z, le združevanje bitov.

Vodilni koeficient zmnožimo z vhodom Z s 15×13 množilnikom in prištejemo začetno vrednost. Ker se zadnjih 12 bitov zmnožka ignorira, se množilnik nekoliko poenostavi. Izhod je 16-bitni (fiksna vejica b3.16).

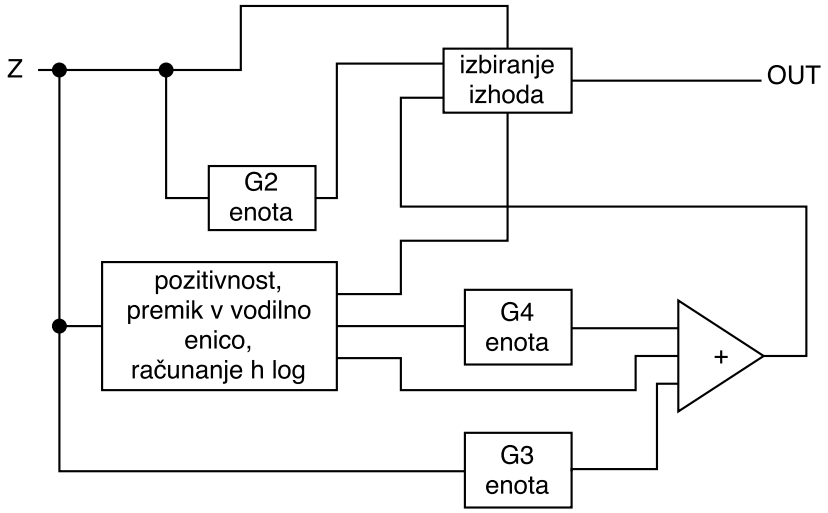
8.1.4 Enota G2G3G4

Vhod Z je 13-bitni (fiksna vejica b3.10), izhod OUT pa je 19-bitni (fiksna vejica b5.14). Enoto s skico na sliki 8.4 sestavljajo naslednje enote.

- Enote G2, G3, G4: evalvacija funkcij g_2, g_3 in g_4.
- LOD - (Leading One Detector) detektor vodilne enice vhoda Z. Sestavljajo ga trije LOD4 (4-bitni detektorji vodilne enice).
- Premik v vodilno enico: če je vhod Z neničelen s kodirnikom in multiplekserji ter detektorjem vodilne enice, premaknemo vhod v vodilno enico.
- Računanje $h \ln 2$: ker h lahko zavzame le 12 vrednosti, zato zmnožek vnaprej izračunamo in zaokrožimo.
- Izbiranje izhoda: če je Z ničelen (indikator pozitivnosti Z-ja je eden izmed izhodov detektorja vodilne enice), je izhod enak LNS(0). Če je Z na $(0, 4)$, je izhod enak izhodu enote G2, sicer pa vsoti izhodov enot G3 in G4 ter zmnožka $h \ln 2$.

8.1.5 Enota G2

Vhod Z je 13-bitni (fiksna vejica b3.10), izhod OUT pa je 17-bitni (fiksna vejica b4.13). Ker imamo samo šest podintervalov, izbiranje koeficientov implementiramo z multiplekserji. Vodilni koeficient je 16-bitni. Začetna vrednost je 10-bitna. Ker je prvih sedem bitov vodilnega koeficienta vedno



Slika 8.4: Skica enote G2G3G4.

enakih „1000000“, bomo shranili le preostale bite. Glede na to, da je veliko bitov ničelnih, bomo število zmnožili v dveh delih. To zmanjša množilnik na 9×13. Tako kot pri G1 se tudi tu množilnik nekoliko poenostavi z ignoriranjem zadnjih 12 bitov zmnožka.

8.1.6 Enota G3

Vhod Z je 12-bitni (fiksna vejica b2.10), izhod OUT pa je 17-bitni (fiksna vejica b3.14). Koeficiente hranimo v 32×27 velikem ROM-u. Vodilni koeficient je 15-bitni. Začetna vrednost je 13-bitna. Ker je prvi bit vodilnega koeficienta vedno enak ena, hranimo samo zadnjih 13 bitov. Preračunavanje naslova ni potrebno, ker je segmentacija enakomerna. Tako kot pri G1 se tudi tu množilnik 15×13 nekoliko poenostavi z ignoriranjem zadnjih 12 bitov zmnožka.

8.1.7 Enota G4

Vhod Z je 12-bitni (fiksna vejica b1.12), izhod OUT pa je 17-bitni (fiksna vejica b3.14). Koeficiente hranimo v 24×27 velikem ROM-u. Vodilni koeficient in začetna vrednost sta 14-bitna, ker pa je prvi bit vodilnega koeficienta

vedno enak ena, hranimo samo zadnjih 13 bitov. Naslov preračunamo na enak način kot pri G1, le da tukaj potrebujemo samo en enobitni primerjalnik, saj se velikost podintervalov spremeni samo enkrat. Tako kot pri G1 se tudi tu množilnik 14×13 nekoliko poenostavi z ignoriranjem zadnjih 12 bitov zmnožka.

8.2 Zakasnitveni časi in poraba LUT-enot

FPGA Spratan3E-500 ima 9313 4-vhodih LUT-ov (Look Up table) oz. 16×1 velikih ROM-ov.

Poraba LUT-ov in zakasnilni časi pri različnih pristopih implementacije evalvacije funkcije g_1 in g_2 so podani s tabelo 8.1. Različni pristopi implementacije:

- Direkten: za vsak vhod hranimo izhod (prekodirnik $8K \times 13$).
- Brez segmentacije: funkcijo na celem intervalu evalviramo z enim polinomom visoke stopnje.
- Enakomerna segmentacija.
- Neenakomerna segmentacija.

	funkcija g_1		funkcija g_2	
pristop	# LUT-ov	maksimalni zakasnitveni čas [ns]	# LUT-ov	maksimalni zakasnitveni čas [ns]
direkten	1898	17.529	-	-
brez seg.	651	97.645	-	-
enakomerna seg.	112	15.080	356	25.570
neenakomerna seg.	118	15.963	340	27.178

Tabela 8.1: Primerjava porabe LUT-ov in zakasnilnih časov.

V tabeli 8.1 vidimo, da pri direktni implementaciji porabimo zelo veliko LUT-ov. Pri pristopu brez segmentacije je zakasnilni čas velik zaradi velikega števila zaporednih množenj. Rezultati pri enakomerni in neenakomerni segmentaciji so zavajajoči, saj optimizator logike izkorišča vsebino asinhronskega ROM. V realni implementaciji bi se uporabil sinhronski ROM, pri katerem bi bila opazna razlika med neenakomerno in enakomerno segmentacijo v porabi LUT-ov.

8.3 Zakasnitveni časi in poraba LUT-enot pri enoti za množenje ter deljenje

Poraba LUT-enot je 34 od 9312 z maksimalnim zakasnilnim časom 10.711 ns. Pri implementaciji pravilnega monoženja z LNS(0) se porabi 55 od 9312 LUT-enot z maksimalnim zakasnilnim časom 12.059 ns.

8.4 Zakasnitveni časi in poraba LUT-enot pri celotni aritmetični enoti za seštevanje in odštevanje

Poraba LUT-enot pri neenakomerni segmentaciji je 585 od 9312 z maksimalnim zakasnilnim časom 39.148 ns. Pri enakomerni segmentaciji pa 596 od 9312 z maksimalnim zakasnilnim časom 38.783 ns.

Poglavje 9

Testi

Testi so bili opravljeni v P_1 z neenakomerno segmentacijo.

9.1 Test seštevanja

Zanima nas relativna napaka pri seštevanju n naključnih predstavljivih števil. Rezultati so podani v tabeli 9.1.

n	povprečna relativna napaka
10	0.0024
100	0.0061
1000	0.0307

Tabela 9.1: Povprečne relativne napake pri seštevanju.

V tabeli 9.1 vidimo, da je rast napake približno $\mathcal{O}(\sqrt{n})$.

9.2 Test – razcep Choleskega

Naj bo matrika $A \in \mathbb{R}^{n\times n}$ pozitivno definitna, potem obstaja razcep Choleskega $A = VV^T$, kjer je V matrika v spodnje trikotni obliki s pozitivnimi

elementi na diagonali [19]. Če je matrika tudi tridiagonalna, velja naslednji algoritem.

Algoritem 3. (Razcep Choleskega za tridiagonalno matriko)

$$V_{jj} = \sqrt{A_{jj} - V_{jj-1}^2}$$

$$V_{j+1j} = \frac{1}{V_{jj}} A_{j+1j}$$

Rezultati so podani v tabeli 9.2.

n	povprečna relativna napaka
8	0.0018
16	0.0026
32	0.0040
64	0.0048
128	0.0059

Tabela 9.2: Povprečna maksimalna relativna napaka pri razcepu Choleskega.

V plavajoči vejici razcep Choleskega lahko predstavlja problem, ker počasne operacije, kot so korenjenje, množenje in deljenje, prevladujejo nad odštevanji oz. seštevanji. V LNS je v smislu počasnih operacij ravno obratno. Problem bi lahko povzročale numerične napake pri seštevanju oz. odštevanju. Rezultati testa v tabeli 9.2 kažejo na to, da je seštevanje in odštevaje v primeru računanja razcepa Choleskega numerično pogojeno.

9.3 Test – kodiranje in dekodiranje JPEG

Kodiranje slike v RGB-formatu v JPEG poteka (v splošnem) v naslednjih korakih [4]:

- Sliko iz RGB-formata pretvorimo v YCbCr-format.
- V formatu YCbCr po potrebi zmanjšujemo barvne komponente (down-sampling).

- Vsako barvno komponento razdelimo na bloke, velike 8×8.
- Vsak blok transformiramo z diskretno kosinusno transformacijo (DCT).
- Transformiranko (frekvenčno območje) po elementih delimo in zaokrožimo na celo število z matriko Q, ki definira kompresijsko razmerje.
- Nad vsemi matrikami skupaj izvedemo kodiranje entropije (lossless compression).

Dekodiranje poteka v obratnem vrstnem redu, s tem da na mesto diskretne kosinuse transformacije (DCT) uporabimo inverzno diskretno kosinusno transformacijo (IDCT).

Kodiranje in dekodiranje sta implementirani v 16-bitnem LNS (v P_1 z neenakomerno segmentacijo), tako da se množenja z matriko, DCT in IDCT izvajajo v LNS, za ostale operacije se pa izvede pretvorba bodisi iz LNS v plavajočo vejico (fiksna vejica) bodisi iz plavajoče vejice (fiksna vejica) v LNS. V našem primeru pri kodiranju ne zmanjšujemo barvne komponente (downsampling), zato nam tudi pri dekodiranju ni treba povečevanje barvnih komponent (upsampling).

Primerjava kodiranja in dekodiranja JPEG med 16-bitnim LNS in dvojno natančnostjo je bila izvedena s sliko 9.1.

Po kodiranju in dekodiranju pri 50 % kompresijskem razmerju dobimo s 16-bitnim LNS sliko 9.2 in s 64-bit FP sliko 9.3.

Povprečna absolutna razlika med originalom in pri implementacij v 16-bitnem LNS je 3.6946, v dvojni natančnosti pa 3.6913. Opazimo, da se kompresirani sliki 9.2 in 9.3 na pogled ne razlikujeta ter da povprečna razlika odstopa zelo malo. Če lahko na hiter (v primerjavi z računanjem DCT, IDCT in množenjem z Q) način izvedemo pretvorbo med številskimi sistemi, je uporaba LNS pri JPEG-kodiranju in dekodiranju smiselna.

Slika 9.1: Originalna vhodna slika v velikosti 256×256 [11].

Slika 9.2: Po kodiranju in dekodiranju v 16-bitnem LNS.

Slika 9.3: Po kodiranju in dekodiranju v 64-bit FP.

Poglavje 10

Zaključek

Za konec lahko povzamemo naslednje ugotovitve glede baze e v LNS. Naš LNS ima velik interval $\approx [-8.9M, 8.9M]$ predstavljivih števil z uporabo samo 16 bitov v primerjavi s polovično natančnostjo. V poglavju 2.2.4 smo pokazali, da je baza e optimalna izbira glede ekonomije bitov [8]. V bazi e je težje pretvarjanje med plavajočo vejico in LNS kot v bazi 2, saj je potrebno dodatno množenje z $\ln 2$ oz. $\log_2 e$ pri računanju $e^z = 2^{z \log_2 e}$ in $\ln x$. Baza e nam ne omogoča implementacije aritmetične enote, ki bi bila boljša od analognega ekvivalenta v plavajoči vejici (napaka pri seštevanju in odštevanju bo vedno večja). V bazi 2 lahko implementiramo aritmetično enoto, ki je BTFP, ampak je težje aproksimirati sumacijsko in diferenčno funkcijo v bazi 2 kot g_1 in g_2 v bazi e.

Pri možnostih za nadaljnje izboljšave je najpomembnejša sinhronizacija celotne 16-bitne aritmetične enote v LNS. To vključuje načrtovanje in implementacijo cevovoda ter uporabo sinhronih bralnih pomnilnikov. Lahko se nadgradi enota z dodajanjem aritmetičnih operacij in funkcij, kot je e^x in potenciranje s 15-bitnim številom v fiksni vejici. Pri aproksimaciji funkcij g_1 in g_2 se lahko izboljša balansiranje napake z boljšo izbiro segmentacije, načinom zaokroževanja koeficientov in zmnožkov. S tem bi zmanjšali velikost množilnikov in število bitov, potrebnih za reprezentacijo koeficientov ter vmesnih rezultatov.

Moje mnenje je, da je implementacija splošnonamenske aritmetične enote v LNS nesmiselna, če bomo vhode in izhode pri reševanju problemov morali pretvarjati med LNS in FP. Če je delež operacij, porabljenih za pretvorbo, zanemarljiv oz. majhen v primerjavi s številom operacij, ki se izvajajo v LNS-aritmetiki, potem je razlog za uporabo take aritmetične enote v kombinaciji s pretvorbami še vedno zadovoljiv.

Literatura

[1] Chebfun. Dosegljivo: http://www.chebfun.org/. [Dostopano: 22. 8. 2017].

[2] Gravity Pipe. Dosegljivo: https://en.wikipedia.org/wiki/Gravity_Pipe. [Dostopano: 16. 2. 2017].

[3] Half-precision floating-point format. Dosegljivo: https://en.wikipedia.org/wiki/Half-precision_floating-point_format. [Dostopano: 3. 3. 2017].

[4] JPEG. Dosegljivo: https://en.wikipedia.org/wiki/JPEG. [Dostopano: 10. 8. 2017].

[5] Kahan summation algorithm. Dosegljivo: https://en.wikipedia.org/wiki/Kahan_summation_algorithm. [Dostopano: 4. 6. 2017].

[6] Logarithmic number system. Dosegljivo: https://en.wikipedia.org/wiki/Logarithmic_number_system. [Dostopano: 3. 3. 2017].

[7] Nexys 2. Dosegljivo: https://reference.digilentinc.com/reference/programmable-logic/nexys-2. [Dostopano: 22. 8. 2017].

[8] Radix economy. Dosegljivo: https://en.wikipedia.org/wiki/Radix_economy. [Dostopano: 23. 5. 2017].

[9] Random walk. Dosegljivo: https://en.wikipedia.org/wiki/Random_walk. [Dostopano: 4. 6. 2017].

[10] The memory format of an IEEE 754 half precision floating point. Dosegljivo: `https://upload.wikimedia.org/wikipedia/commons/2/21/IEEE_754r_Half_Floating_Point_Format.svg`. [Dostopano: 22. 8. 2017].

[11] Wasserkuppe Gersfeld. Dosegljivo: `https://media-cdn.tripadvisor.com/media/photo-s/04/10/f4/63/wasserkuppe-gersfeld.jpg`. [Dostopano: 10. 8. 2017].

[12] Encyclopædia Britannica Eleventh Edition. Cambridge University Press, 1911.

[13] Charles H. Cotter. Gaussian Logarithms and Navigation. *Journal of Navigation*, pages 569–572, 1971.

[14] Nicholas J. Higham. The Accuracy of Floating Point Summation. *SIAM Journal on Scientific Computing*, pages 783–799, 1993.

[15] IEEE. IEEE Standard for Binary Floating-Point Arithmetic. *ANSI/IEEE Std 754-1985*, 1985.

[16] IEEE. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008.

[17] N. G. Kingsbury and P. J. W. Rayner. Digital filtering using logarithmic arithmetic. *Electronics Letters*, pages 56–58, 1971.

[18] Kozak, J. *Numerična analiza*. Matematika – fizika: zbirka univerzitetnih učbenikov in monografij. DMFA – založništvo, 2008.

[19] B. Plestenjak. *Razširjen uvod v numerične metode*. Matematika – fizika: zbirka univerzitetnih učbenikov in monografij. DMFA – založništvo, 2015.

[20] Toshiyuki Fukushige and Piet Hut and Jun Makino. High-Performance Special-Purpose Computers in Science. *IEEE*, 1999.

[21] Vanderbauwhede, W. and Benkrid, K. *High-Performance Computing Using FPGAs.* SpringerLink : Bücher. Springer New York, 2013.